QR FACTORIZATION OVER TUNABLE PROCESSOR GRIDS

BY

EDWARD HUTTER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Electrical and Computer Engineering
in the College of Engineering of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Edgar Solomonik

Abstract

The increasing complexity of modern computer architectures has greatly influenced algorithm design. Algorithm performance on these architectures is now determined by the movement of data. Therefore, modern algorithms should prioritize minimizing communication. In this work, we present a new parallel QR factorization algorithm solved over a tunable processor grid in a distributed memory environment. The processor grid can be tuned between one and three dimensions, resulting in tradeoffs in the asymptotic costs of synchronization, horizontal bandwidth, flop count, and memory footprint. This parallel algorithm is the first to efficiently extend the Cholesky-QR2 algorithm to matrices with an arbitrary number of rows and columns. Along its critical path of execution on P processors, our tunable algorithm improves upon the horizontal bandwidth cost of the existing CholeskyQR2 algorithm by up to a factor of c^2 when solved over a $c \times d \times c$ processor grid subject to $P = c^2 d$ and $c \in [1, P^{\frac{1}{3}}]$. The costs attained by our algorithm are asymptotically equivalent to state of the art QR factorization algorithms that have yet to be implemented. We argue that ours achieves better practicality and flexibility while still attaining minimal communication.

Subject keywords — communication cost; parallel numerical algorithms; least squares problems; QR factorization

To my parents, for their love and support.

Acknowledgments

I want to thank my advisor, Prof. Edgar Solomonik, for his enthusiastic involvement in this project. Professor Solomonik has introduced me to the field of communication-avoiding algorithms and has provided both theoretical insight and practical knowledge invaluable to my progress. I look forward to working with him as I pursue my PhD in Computer Science in the Fall.

Contents

List of Terms

Latency/Synchronization cost — elapsed time between request and reception of the first byte of data in a message.

Vertical bandwidth cost — cost of moving a byte between levels of a memory hierarchy multiplied by the number of bytes moved.

Horizontal bandwidth cost — cost of moving a byte among processors over a network multiplied by the number of bytes moved.

Communication cost — linear combination of horizontal bandwidth cost and synchronization cost.

Memory footprint — number of bytes utilized by some subset of an algorithm on a single processor.

Flop cost — cost of a floating-point operation with register-resident data multiplied by the number of such operations.

Critical path — most expensive chain of dependent execution of an algorithm that can be represented as a path along some subset of the directed acyclic graph describing said algorithm.

Numerical stability — ability of an algorithm to suppress input and approximation errors.

Chapter 1

Introduction

1.1 Background

The end of Moore's law and the rise in access to multicore and distributed-memory machines has sparked a change in the way algorithms must be designed. Current trends show that the on-chip overall flop rate is increasing exponentially faster than the memory bandwidth, and the memory bandwidth is increasing exponentially faster than the rate at which latency is decreasing. This trend is expected to continue into the foreseeable future for both horizontal and vertical movement [1]. Current algorithms thus will be spending an increasingly longer time moving data from outer memory levels to the CPU than computing results with that data. In response, algorithms are being redesigned to prioritize minimizing communication, even at the expense of a higher flop count.

In general, the success of numerical linear algebra algorithms is determined by many factors, including numerical stability, scalability, the costs of memory footprint, flop count, synchronization, and communication, among many others. Library implementations of widely used linear algebra primitives in BLAS, LAPACK, and SCALAPACK tune each algorithm for specific architectures in order to exploit differences in cache size, number of cache levels, and superscalar and vectorization capabilities to name a few [2]. In addition, these algorithms take advantage of specific matrix layouts and use techniques such as loop blocking and unrolling to minimize cache misses. Designing and implementing just a single numerical linear algebra algorithm requires careful attention to many details.

Parallel dense linear algebra algorithms can be categorized based on the arrangement of processing elements. Two-dimensional (2D) and three-dimensional (3D) parallel algorithms are solved over 2D and 3D grids, respectively, and have tradeoffs involving synchronization cost, communication cost, and memory footprint. These algorithms must balance the workload and reduce interprocessor communication, while employing a minimal amount of memory. It is imperative that the length of the algorithms' critical paths be minimized, im-

plying maximal contributions by all processors on every step of the algorithm. In this thesis, we investigate and implement a few existing numerical linear algebra algorithms solved over a collection of processor grids. We use these algorithms as building blocks when presenting our tunable QR factorization algorithm.

1.2 Motivation

Tall and skinny matrices are ubiquitous in many applications within the fields of scientific computing and machine learning. They represent repeated observations of some phenomena, where the number of observations vastly outnumbers the number of unknown linear system coefficients. Solutions of such systems will rarely lie in the column space of the corresponding matrix. The goal is then to find a set of coefficients that minimizes the 2-norm of the residual. QR factorizations are often the key step to solving these least squares problems.

A few papers provided key insight into the development of our 3D and tunable QR factorization algorithms. Recent work on parameterizing 2D and 3D algorithms by memory size has been proven to reduce communication [3]. These 2.5D algorithms take advantage of extra memory and a tunable processor grid to achieve costs that interpolate between those of known 2D and 3D algorithms. Applications for 2.5D dense linear algebra algorithms include graph algorithms such as All Pairs Shortest Paths (APSP) [4]. These divide and conquer APSP algorithms follow a critical path similar to our recursive Cholesky algorithm, conquering the square matrix one diagonal at a time.

Much focus has been given to theoretical analysis of the communication cost of matrix multiplication and Gaussian elimination algorithms. We present a simple adaptation of the algorithm therein [5] to Cholesky factorization. Tradeoffs between synchronization and bandwidth are investigated using a tuneable parameter, α, representing the depth of recursion. This parameter thus determines the size of the diagonal submatrices. We focus on minimizing bandwidth cost given unbounded memory so we choose to use the value $\alpha = \frac{2}{3}$.

The main motivation of our work comes from CholeskyQR2, a recent algorithm for tall and skinny matrices that substantially improves upon the numerical instability of CholeskyQR by performing CholeskyQR twice [6]. The advantages of this algorithm lie in its practicality. It requires only matrix multiplications and Cholesky factorizations. While easy to parallelize and implement, its current design limits its scability. Its communication cost has been investigated [7] as well as its numerical stability [8]. It achieves twice the communication

cost of state of the art QR factorization algorithms for tall and skinny matrices such as TSQR [9] and achieves ideal numerical stability for any matrix with a condition number of $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$, where ϵ is the machine epsilon. Consequently, the algorithm is of high practical interest for sufficiently well-conditioned least squares problems. We aim at extending its reach to matrices of an arbitrary size.

Of the QR algorithms that achieve asymptotically minimal communication, none have been implemented in practice perhaps due to the impractical complexity of running each on large-scale distributed memory machines. One such algorithm uses a 3D design that makes it communication efficient for square matrices [10]. Recent work has extended the algorithm to rectangular matrices by using it as a subroutine in the TSQR algorithm for tall and skinny matrices with Tiskin's algorithm [11]. This algorithm achieves optimally minimal communication for rectangular matrices.

1.3 Contributions

The common denominator illustrated above is that no current QR factorization algorithm achieves both practicality and optimally minimal communication cost. Algorithms achieving such cost have never been implemented in practice because of their complexity. Simpler algorithms such as CholeskyQR2 do not scale for matrices of an arbitrary size. The novelty of our tunable CholeskyQR2 algorithm lies in its ability to accomplish both objectives. It relies upon a tunable processor grid to ensure optimally minimal communication for matrices of any size while being relatively easy to implement. We provide a detailed specification and cost analysis of the proposed algorithm.

Chapter 2

Algorithm Design

In order to understand our tunable Cholesky-QR2 QR factorization algorithm, it is necessary to understand its building blocks, namely collective communication, matrix multiplication, and Cholesky factorization. A wide range of choices is available for each building block, so we give specifics of each chosen algorithm and provide a thorough analysis on the design choices made for the tunable Cholesky-QR2 algorithm.

2.1 Collective Communication

Collective communication serves as an efficient way to move data among processors over some subset of a processor grid. We partition the processor grid into slices, rows, and columns, among other subdivisions, by splitting communicators into subcommunicators. We define a 3D processor grid Π containing P processors. $\Pi[i,j,k]$ uniquely identifies every processor in the grid, where each of the x, y, and z dimensions are of size $P^{\frac{1}{3}}$ and $i,j,k \in [0, P^{\frac{1}{3}}-1]$. Π can be split into 2D slices such as $\Pi[:,:,k]$, row communicators such as $\Pi[:,j,k]$, column communicators such as $\Pi[i,:,k]$, and depth communicators such as $\Pi[i,j,:]$. See figure 2.1 for a diagram of Π.

AllGather, AllReduce, and Broadcast are collectives used heavily in the multiplication and factorization algorithms explored below. As these are well known, we give only the function signature and a brief description of each. We assume butterfly network collectives for analysis.

- **Bcast**$(A, B, n, \Pi[:,j,k])$ — root processor n distributes local array A to every processor in $\Pi[:,j,k]$ as local array B.
- **AllReduce**$(A, B, \Pi[i,:,k])$ — all processors in $\Pi[i,:,k]$ contribute local arrays A to an element-wise reduction onto some root. The reduced array is then broadcasted into local array B.

- **AllGather**$(A, B, \Pi[i, j, :])$ — all processors in $\Pi[i, j, :]$ contribute local arrays A to a concatenation onto some root. The concatenated array is then broadcasted into local array B.

2.2 Matrix Multiplication

Matrix multiplication $C = AB$ over Π and other cubic partitions of Π is an important building block for the 3D and tunable Cholesky-QR2 algorithms presented below. We use a variant of 3D matrix multiplication (which we refer to as 3D SUMMA) that achieves asymptotically optimal communication cost over a 3D processor grid [12, 13, 14, 15, 16, 17]. Our algorithm MatrixMultiplication3D is similar to known algorithm but includes a few minor modifications. First, B is not distributed across $\Pi[P^{\frac{2}{3}}-1, :, :]$ and is instead distributed across $\Pi[:, :, 0]$ with A. Second, instead of distributing matrix C across $\Pi[:, 0, :]$, we AllReduce C onto $\Pi[:, :, k], \forall k \in [0, P^{\frac{1}{3}} - 1]$ so that each xy slice holds a distributed copy. These differences are motivated by the need for C to be replicated over Π in our new algorithms. 3D SUMMA would require an unnecessary broadcast from $\Pi[:, 0, :]$ to every other xz slice of Π. Figure 2.1 illustrates the communication patterns of the algorithm and includes a figure detailing the cyclic distribution of matrices A, B, and C onto $\Pi[:, :, 0]$. This cyclic distribution is required by our 3D Cholesky factorization algorithm detailed below. Note that an initial broadcast must occur along dimension z which is not pictured in Figure 2.1. See Algorithm 1 for specific details.

Algorithm 1 $[C] \leftarrow$ **MatrixMultiplication3D** $(A, B, m, n, k, \Pi, i, j, k)$

Require: Π has P processors arranged in a 3D grid. Matrices A and B are replicated on $\Pi[:, :, k], \forall k \in [0, P^{\frac{1}{3}} - 1]$. Each processor $\Pi[i, j, k]$ owns a cyclic partition of $m \times n$ matrix A and $n \times k$ matrix B. We call these local matrices A_{ij} and B_{ij}, respectively. These matrix partitions are condensed into 1D row-major arrays of size $\frac{mn}{P^{\frac{2}{3}}}$ and $\frac{nk}{P^{\frac{2}{3}}}$, respectively. Let X, Y, and Z be temporary arrays with the same distribution as A and B.

1: **Bcast**$(A_{kj}, X_{ij}, k, \Pi[:, j, k])$ ▷ Broadcast from root k across row i
2: **Bcast**$(B_{ik}, Y_{ij}, k, \Pi[i, :, k])$ ▷ Broadcast from root k along column j
3: $Z_{ij} \leftarrow$**seq-MM**$\left(X_{ij}, Y_{ij}, \frac{n}{P^{\frac{1}{3}}}\right)$
4: **Allreduce**$(Z_{ij}, C_{ij}, \Pi[i, j, :])$ ▷ AllReduce along the depth of 3D grid

Ensure: $C = AB$, where C is $m \times k$ and distributed the same way as A and B.

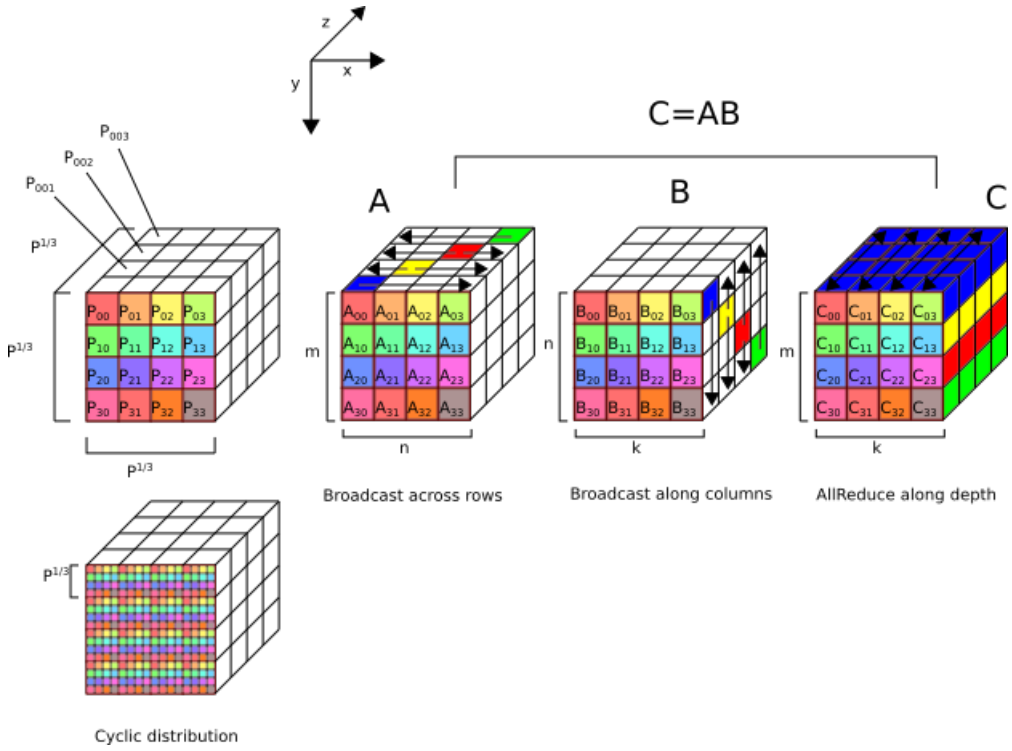

Figure 2.1: Illustration of the layout of matrices A, B, C over Π and the communication required in MatrixMultiplication3D

2.3 Cholesky Factorization

Assuming A is a dense, symmetric positive definite matrix of dimension n, the factorization $A = LL^T$ can be expanded into matrix multiplication of submatrices of dimension $\frac{n}{2}$ [5].

$$\begin{bmatrix} A_{11} & \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ & L_{22}^T \end{bmatrix}$$

$$\begin{aligned} A_{11} &= L_{11}L_{11}^T \\ A_{21} &= L_{21}L_{11}^T \\ A_{22} &= L_{21}L_{21}^T + L_{22}L_{22}^T \end{aligned}$$

Rewriting these equations gives a recursive definition for $L = \text{Cholesky}(A)$.

$$\begin{aligned} L_{11} &= \text{Cholesky}(A_{11}) \\ L_{21} &= A_{21}L_{11}^{-T} \\ L_{22} &= \text{Cholesky}\left(A_{22} - L_{21}L_{21}^T\right) \end{aligned}$$

To complete the recursive definition of the Cholesky factorization, a recursive definition for L^{-1} must be derived. Like before, the factorization $I_n = LL^{-1}$ gets expanded into matrix multiplication of submatrices of dimension $\frac{n}{2}$.

$$\begin{bmatrix} I_{\frac{n}{2}} & 0 \\ 0 & I_{\frac{n}{2}} \end{bmatrix} = \begin{bmatrix} L_{11} & \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^{-1} & \\ L_{21}^{-1} & L_{22}^{-1} \end{bmatrix}$$
$$I_{\frac{n}{2}} = L_{11}L_{11}^{-1}$$
$$I_{\frac{n}{2}} = L_{22}L_{22}^{-1}$$
$$0_{\frac{n}{2}} = L_{21}L_{11}^{-1} + L_{22}L_{21}^{-1}$$

Rewriting these equations gives a recursive definition for $L^{-1} = \text{Inverse}\,(L)$.

$$L_{11}^{-1} = \text{Inverse}\,(L_{11})$$
$$L_{21}^{-1} = -L_{22}^{-1}L_{21}L_{11}^{-1}$$
$$L_{22}^{-1} = \text{Inverse}\,(L_{22})$$

We embed the two recursive definitions and arrive at an algorithm to solve for both the Cholesky factorization of A and the triangular inverse of L. Note that the addition of solving for L^{-1} adds only two extra matrix multiplicatons at each recursive level to the recursive definition for $A = LL^T$, thus achieving the same asymptotic cost. If L^{-1} were to be solved recursively at each level, the communication cost would incur an extra logarithmic factor. We address the missing base case in the cost analysis derivation in chapter 3.

$$\begin{bmatrix} L & L^{-1} \end{bmatrix} = \text{CholeskyInverse}\,(A)$$

$$\begin{bmatrix} L_{11} & L_{11}^{-1} \end{bmatrix} = \text{CholeskyInverse}\,(A_{11})$$
$$L_{21} = A_{21}L_{11}^{-T}$$
$$\begin{bmatrix} L_{22} & L_{22}^{-1} \end{bmatrix} = \text{CholeskyInverse}\left(A_{22} - L_{21}L_{21}^T\right)$$

$$L = \begin{bmatrix} L_{11} & \\ L_{21} & L_{22} \end{bmatrix}$$
$$L^{-1} = \begin{bmatrix} L_{11}^{-1} & \\ -L_{22}^{-1}L_{21}L_{11}^{-1} & L_{22}^{-1} \end{bmatrix}$$

We incorporate two matrix transposes at each recursive level to take into account L_{11}^{-T} and L_{21}^T needed in the equations above. Processor $\Pi[i,j,k], \forall k \in [0, P^{\frac{1}{3}} - 1]$ must send its local data to $\Pi[j,i,k]$ to transpose the matrix globally. A local transpose without horizontal communication will yield an incorrect distributed transpose as illustrated in Figure 2.2.

A cyclic distribution of the matrices among processors $\Pi[:,:,k], \forall k \in [0, P^{\frac{1}{3}} - 1]$ is chosen to utilize every processor in the recursive calls performed on submatrices. Upon reaching the base case where matrix dimension $n = n_o$, the square region of the matrix is scattered over the $P^{\frac{2}{3}}$ processors encompassing $\Pi[:,:,k], \forall k \in [0, P^{\frac{1}{3}} - 1]$ and an AllGather must be performed to load the region onto each. As discussed in chapter 3, $n_o^2 = \frac{n^2}{P^{\frac{4}{3}}}$, so the memory footprint produced by this AllGather has no asymptotic affect. After all P processors perform a Cholesky factorization and triangular inverse, each stores only the data it owns according to the cyclic rule. See Algorithm 2 for full details.

Figure 2.3 illustrates the critical path of the algorithm. The base case factors each square block of dimension n_o along the diagonal of A in order from left to right. Upon solving for L_{11} and L_{11}^{-1}, the path travels downward to calculate L_{21}. After performing the necessary transposes and matrix multiplications, the next diagonal block is conquered and we attain L_{22} and L_{22}^{-1}. The last steps of every recursive level involve two matrix multiplications needed to solve for L_{21}^{-1}. The algorithm completes after every diagonal block has been factored.

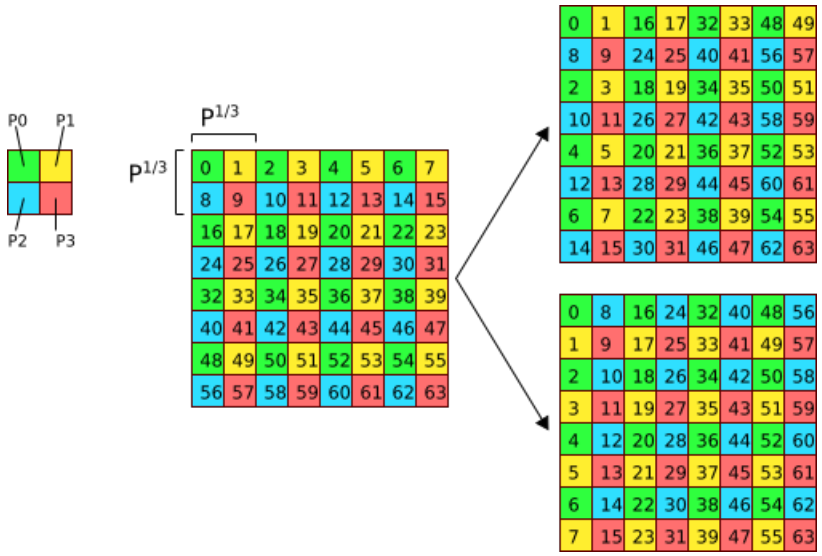

Figure 2.2: Illustration of the need for communication when performing the transpose in steps 7 and 9 of Algorithm 2. The top picture performs the transpose locally and the bottom picture uses point-to-point communication and a local transposition.

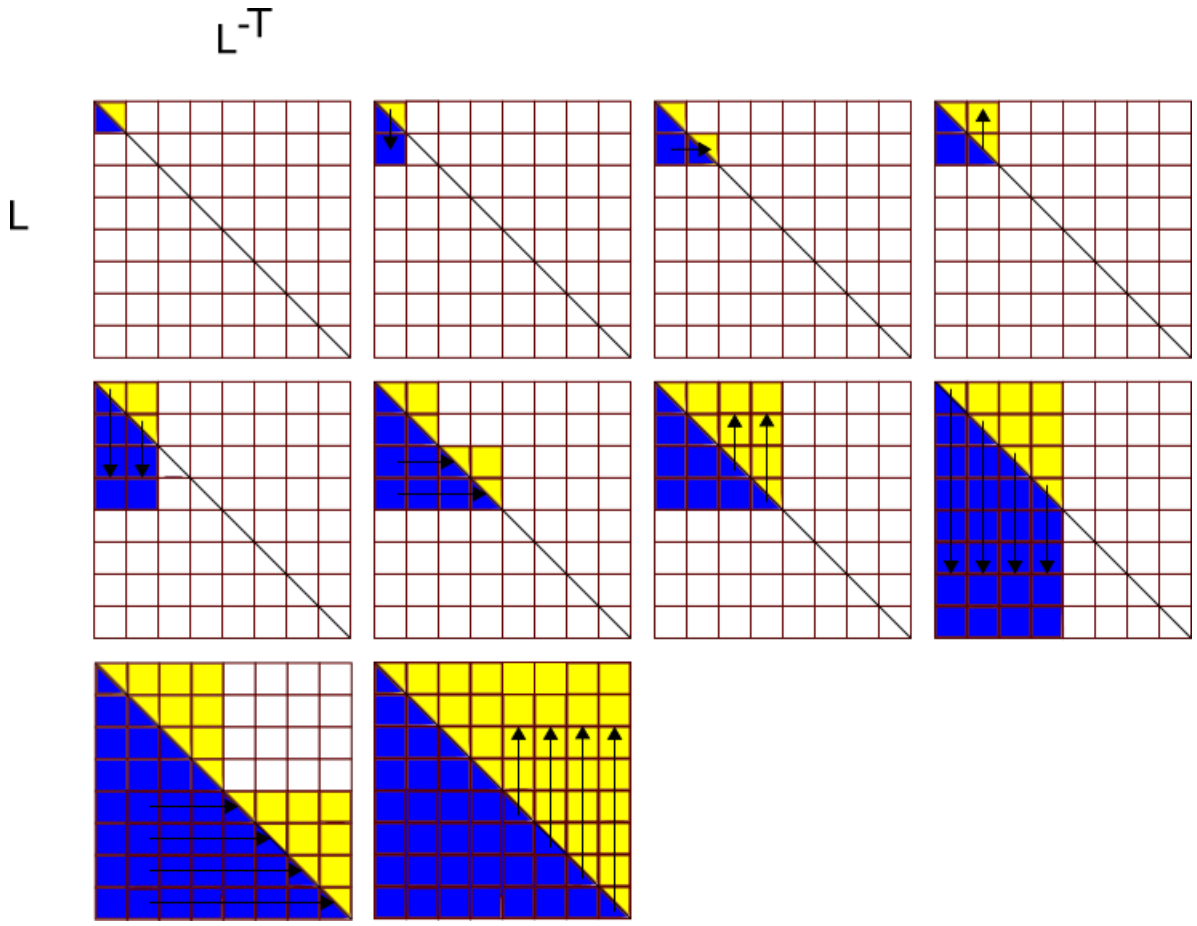

Figure 2.3: The critical path of Algorithm 2 is shown after first recursing to the top-left most diagonal in the matrix.

Algorithm 2 $[L, L^{-1}] \leftarrow$ **CholeskyFactorization3D** $(A, n, n_o, \Pi, i, j, k)$

Require: Π has P processors arranged in a 3D grid. Matrix A is of dimension n, symmetric, and positive definite. A is replicated on $\Pi[:, :, k], \forall k \in [0, P^{\frac{1}{3}} - 1]$. Each processor $\Pi[i, j, k]$ owns a cyclic partition of A known as A_{ij}. A_{ij} is packed into a 1D array of size $\frac{\left(\frac{n}{P^{\frac{1}{3}}}\right)\left(\frac{n}{P^{\frac{1}{3}}}+1\right)}{2}$. Let n_o be the matrix dimension in which we call the base case. Let $TopLeft$ and $BottomRight$ be lower triangular portions of dimension $\frac{n}{2}$ square submatrices in the upper-left quadrant and lower-right quadrants of a matrix, respectively. Let $BottomLeft$ be the square submatrix in the lower-left quadrant. Let T, W, X, Y, and Z be temporary arrays, distributed the same way as A.

1: **if** $n = n_o$ **then**
2: **AllGather** $(A_{ij}, T_{ij}, \Pi[:, :, k])$
3: $L_{ij} \leftarrow$ **seq-Cholesky** (T_{ij}, n)
4: $L_{ij}^{-1} \leftarrow$ **seq-TriInv** (L_{ij}, n)
5: **else**
6: $L[TopLeft], L^{-1}[TopLeft] \leftarrow$ **CholeskyFactorization3D** $\left(A[TopLeft], \frac{n}{2}, n_o, \Pi, i, j, k\right)$
7: $W_{ij} \leftarrow$ **Transpose** $\left(L_{ij}^{-1}[TopLeft], \Pi[j, i, k]\right)$
8: $L[BottomLeft] \leftarrow$ **MatrixMultiply3D** $\left(A[BottomLeft], W^T, \frac{n}{2}, \frac{n}{2}, \frac{n}{2}, \Pi, i, j, k\right)$
9: $X_{ij} \leftarrow$ **Transpose** $(L_{ij}[BottomLeft], \Pi[j, i, k])$
10: $Y \leftarrow$ **MatrixMultiply3D** $\left(L[BottomLeft], X^T, \frac{n}{2}, \frac{n}{2}, \frac{n}{2}, \Pi, i, j, k\right)$
11: $Z_{ij} \leftarrow$ **seq-Subtract** $\left(A_{ij}[BottomRight], Y_{ij}, \frac{n}{2}, \frac{n}{2}\right)$
12: $L[BottomRight], L^{-1}[BottomRight] \leftarrow$ **CholeskyFactorization3D** $\left(Z, \frac{n}{2}, n_o, \Pi, i, j, k\right)$
13: $Y \leftarrow$ **MatrixMultiply3D** $\left(L[BottomLeft], L^{-1}[TopLeft], \frac{n}{2}, \frac{n}{2}, \frac{n}{2}, \Pi, i, j, k\right)$
14: $W \leftarrow (-1) \cdot L^{-1}[BottomRight]$
15: $L^{-1}[BottomLeft] \leftarrow$ **MatrixMultiply3D** $\left(W, Y, \frac{n}{2}, \frac{n}{2}, \frac{n}{2}, \Pi, i, j, k\right)$

Ensure: $A = LL^T$, $L^{-1} = (L)^{-1}$, where matrices L and L^{-1} are distributed the same way as A.

2.4 QR Factorization

QR factorization decomposes an $m \times n$ matrix A into matrices Q and R such that $A = QR$. We focus on the case when $m \geq n$ and Q and R are the results of a reduced QR factorization. In this case, Q is $m \times n$ with orthonormal columns and R is $n \times n$ and upper-triangular. See Figure 2.4 for an illustration of matrices A, Q, and R. Algorithms 3 and 4 give pseudocode for the sequential CholeskyQR2 algorithm. It is composed of matrix multiplications and Cholesky factorizations and unlike other QR factorization algorithms, does not require explicit QR factorizations [6]. Using the building block algorithms explored above, we seek to extend the existing parallel CholeskyQR2 algorithm given in Algorithms 5 and 6 to handle an arbitrary number of rows and columns.

Algorithm 3 $[Q, R] \leftarrow$ **CholeskyQR** (A, m, n)

Require: A is $m \times n$

1: $W \leftarrow$ **seq-Syrk** (A, m, n)
2: $R^T \leftarrow$ **seq-Cholesky** (W, n)
3: $Q \leftarrow$ **seq-MM** (A, R^{-1}, m, n, n)

Ensure: $A = QR$, where Q is $m \times n$ orthogonal, R is $n \times n$ upper triangular

Algorithm 4 $[Q, R] \leftarrow$ **CholeskyQR2** (A, m, n)

Require: A is $m \times n$

1: $Q_1, R_1 \leftarrow$ **CholeskyQR** (A)
2: $Q, R_2 \leftarrow$ **CholeskyQR** (Q_1)
3: $R \leftarrow$ **seq-MM**(R_2, R_1, n, n, n)

Ensure: $A = QR$, where Q is $m \times n$ orthogonal, R is $n \times n$ upper triangular

The existing parallel CholeskyQR2 algorithm is solved over 1D processor grid Π_1 [6]. It partitions the $m \times n$ matrix A into P rectangular chunks of size $\frac{m}{P} \times n$. The motivation behind this parallelization strategy lies in minimal required communication. Each processor can perform a sequential symmetric rank-$\frac{m}{P}$ update (syrk) with its partition of A, resulting in $n \times n$ matrix $B_p = A_p^T A_p, \forall p \in [0, P-1]$. As with any matrix multiplication operation, $B = \sum_{p=0}^{P-1} B_p$ is built via repeated scaling down columns and summing across rows. 1D

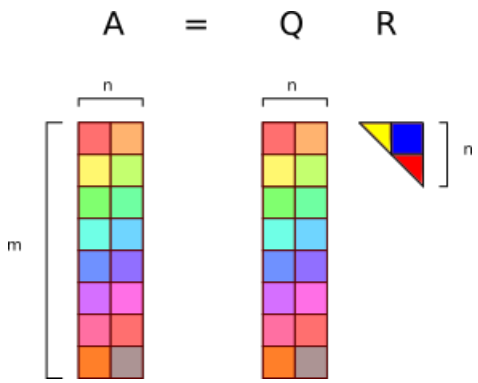

Figure 2.4: Description of the dimensions of matrices Q and R when A is tall and skinny.

parallelization allows each processor to solve its chunk and contribute it to the sum across rows via an AllReduce. Note that each processor need only contribute the upper-triangle of symmetric matrix B_p.

B is now replicated along $\Pi_1[p], \forall p \in [0, P-1]$. Each processor performs a sequential Cholesky factorization with its copy of B and solves for R^{-T}. Finally, because Q is distributed in the same manner as A, horizontal communication is not required and $\Pi_1[p], p \in [0, P-1]$ can solve for Q_p with A_p and its copy of R^{-1}. See Figure 2.5 and Algorithm 5 for finer details.

CholeskyQR2_1D calls CholeskyQR_1D twice to solve for Q as shown in Algorithms 4 and 6. Each processor $\Pi[p], \forall p \in [0, P-1]$ can solve for $R \leftarrow R_2 R_1$ sequentially. This algorithm ensures that Q is distributed the same as A and R is stored on every processor. See Algorithm 6.

To be efficient in practice, n should be small enough to make the AllReduce feasible under given memory constraints. It is most efficient when $m \gg n$. For any given $m \times n$ matrix A, the size and shapes of the local recangular blocks can be tuned. We give a cost analysis in Chapter 3. In total, CholeskyQR2_1D can only be applied to extremeley overdetermined matrices. The CholeskyQR2_3D and CholeskyQR2_Tunable algorithms expand CholeskyQR2 to handle matrices of any size.

Algorithm 5 $[Q, R] \leftarrow$ **CholeskyQR_1D** (A, m, n, Π_1, p)

Require: Π_1 has P processors arranged in a 1D grid. Each processor $\Pi_1[p]$ owns a (cyclic) blocked partition of $m \times n$ input matrix A known as A_p. A_p is packed into a 1D array of size $\frac{mn}{P}$, where it owns a rectangular piece of size $\frac{m}{P} \times n$. Let X and Y be temporary arrays.

1: $X_p \leftarrow$ **seq-Syrk** (A_p, m, n) ▷ $X_p \leftarrow A_p^T A_p$
2: **AllReduce** (X_p, Y_p, Π_1) ▷ $Y \leftarrow A^T A$
3: $R^T \leftarrow$ **seq-Cholesky** (Y, n) ▷ $Y = R^T R$
4: $R^{-T} \leftarrow$ **seq-TriInv** $\left(R^T, n\right)$ ▷ $R^{-T} \leftarrow \left(R^T\right)^{-1}$
5: $Q_p \leftarrow$ **seq-MM** $\left(A_p, R^{-1}, m, n, n\right)$ ▷ $Q \leftarrow AR^{-1}$

Ensure: $A = QR$, where Q is distributed the same as A, R is an upper triangular matrix of dimension n owned locally by every processor and packed into a 1D array of size $\frac{n(n+1)}{2}$.

Algorithm 6 $[Q, R] \leftarrow$ **CholeskyQR2_1D** (A, m, n, Π_1, p)

Require: Same requirements as Algorithm 3.

1: $X, Y \leftarrow$ **CholeskyQR_1D** (A, m, n, Π_1, p)
2: $Q, Z \leftarrow$ **CholeskyQR_1D** (X, m, n, Π_1, p)
3: $R \leftarrow$ **seq-MM** (Z, Y, n, n, n)

Ensure: Same requirements as Algorithm 3.

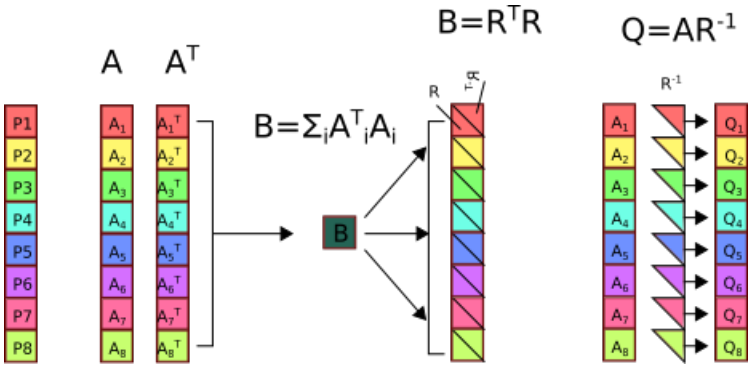

Figure 2.5: Illustration of each step in the existing parallel CholeskyQR algorithm.

We present a new parallel CholeskyQR2 algorithm that factors A over 3D processor grid Π as defined in chapter 2.1. A is distributed across $\Pi[:,:,k], \forall k \in [0, P^{\frac{1}{3}}-1]$ and is partitioned into rectangular chunks of size $\frac{m}{P^{\frac{1}{3}}} \times \frac{n}{P^{\frac{1}{3}}}$. In order to distribute $B = A^T A$ across $\Pi[:,:,k], \forall k \in [0, P^{\frac{1}{3}}-1]$, $\Pi[i,j,k], \forall i,j,k \in [0, P^{\frac{1}{3}}-1]$ must receive submatrix column $A_{:,i}$ from $\Pi[i,:,k]$ and submatrix row $A^T_{j,:}$ from $\Pi[:,j,k]$. Submatrix row $A^T_{j,:}$ is equivalent to submatrix column $A_{:,j}$. Therefore, $\Pi[i,j,k], \forall i,j,k \in [0, P^{\frac{1}{3}}-1]$ can solve for $B_{j,i}$ after communicating two submatrix columns. This can be accomplished efficiently by taking advantage of the 3D grid similar to MatrixMultiplication3D.

As illustrated in Figure 2.6, $\Pi[k,j,k]$ broadcasts its A_{jk} along $\Pi[:,j,k], \forall j,k \in [0, P^{\frac{1}{3}}-1]$. Sequential matrix multiplication is then performed with the received data. Similar to how the AllReduce in CholeskyQR2_1D accomplished summing the rows efficiently in parallel, a reduction along $\Pi[i,:,k]$ onto root $k, \forall i,k \in [0, P^{\frac{1}{3}}-1]$ sums the rows of $B_{j,:}$. Note that an AllReduce could have been used at the same cost but processors $\Pi[i,z,k], \forall i,k \in [0, P^{\frac{1}{3}}-1], z \neq k$ would have no use for the received data. Each xy slice of the grid now owns a different row partition of B. These two communication steps allow Π to conquer each submatrix row $B_{j,:}$ simultaneously. A broadcast along $\Pi[i,j,:], \forall i,j \in [0, P^{\frac{1}{3}}-1]$ ensures that $\Pi[:,:,k], \forall k \in [0, P^{\frac{1}{3}}-1]$ owns a copy of B.

Now that B is distributed the same as A, CholeskyFactorization3D can solve for R^T over Π. Finally, MatrixMultiplication3D solves for Q, which is again distributed the same as A. See Algorithm 7.

CholeskyQR2_3D calls CholeskyQR_3D twice to solve for Q as shown in Algorithms 4 and 8. MatrixMultiplication3D is invoked over Π to solve $R \leftarrow R_2 R_1$. This algorithm ensures that Q and R are distributed the same as A. See Algorithm 8. A cost analysis is presented in Chapter 3.

For problems like QR factorization that solve or factor rectangular matrices, tunable processor grids have several advantages over static grids. They can act as shapeshifters, able to imitate the shape of the matrix and tune themselves to optimize certain parameters such as memory size and horizontal communication. Algorithms written for such grids can

Algorithm 7 $[Q, R] \leftarrow$ **CholeskyQR_3D** (A, m, n, Π, i, j, k)

Require: Π has P processors arranged in a 3D grid. A is $m \times n$ and is replicated on $\Pi[:, :, k], \forall k \in [0, P^{\frac{1}{3}} - 1]$. Each processor $\Pi[i, j, k]$ owns a (cyclic) blocked partition of A known as A_{ji}. A_{ji} is packed into a 1D array of size $\frac{mn}{P^{\frac{2}{3}}}$, where it owns a rectangular piece of size $\frac{m}{P^{\frac{1}{3}}} \times \frac{n}{P^{\frac{1}{3}}}$. Let W, X, Y, Z, and R^{-1} be temporary arrays distributed the same as A.

1: **Bcast** $(A_{jk}, W_{ji}, k, \Pi[:, j, k])$ ▷ Broadcast from root k across row i
2: $X_{ji} \leftarrow$ **seq-MM** $\left(W_{ji}^T, A_{ji}, \frac{n}{P^{\frac{1}{3}}}, \frac{m}{P^{\frac{1}{3}}}, \frac{n}{P^{\frac{1}{3}}}\right)$
3: **Reduce** $(X_{ji}, Y_{ki}, k, \Pi[i, :, k])$ ▷ Reduce along each column to root k
4: **Bcast** $(Y_{ki}, Z_{ji}, k, \Pi[i, j, :])$ ▷ Every 2D slice owns same matrix $B = A^T$
5: $R^T, R^{-T} \leftarrow$ **CholeskyFactorization3D** $\left(Z, n, \frac{n}{P^{\frac{2}{3}}}, \Pi, i, j, k\right)$
6: $Q \leftarrow$ **MatrixMultiplication3D** $\left(A, R^{-1}, m, n, n, \Pi, i, j, k\right)$

Ensure: $A = QR$, where Q and R are distributed the same as A. Q is $m \times n$ and R is an upper triangular matrix of dimension n.

Algorithm 8 $[Q, R] \leftarrow$ **CholeskyQR2_3D** (A, m, n, Π, i, j, k)

Require: Same requirements as Algorithm 5.

1: $X, Y \leftarrow$ **CholeskyQR_3D** (A, m, n, Π, i, j, k)
2: $Q, Z \leftarrow$ **CholeskyQR_3D** (X, m, n, Π, i, j, k)
3: $R \leftarrow$ **MatrixMultiplication3D** $(Z, Y, n, n, n, \Pi, i, j, k)$

Ensure: Same requirements as Algorithm 5.

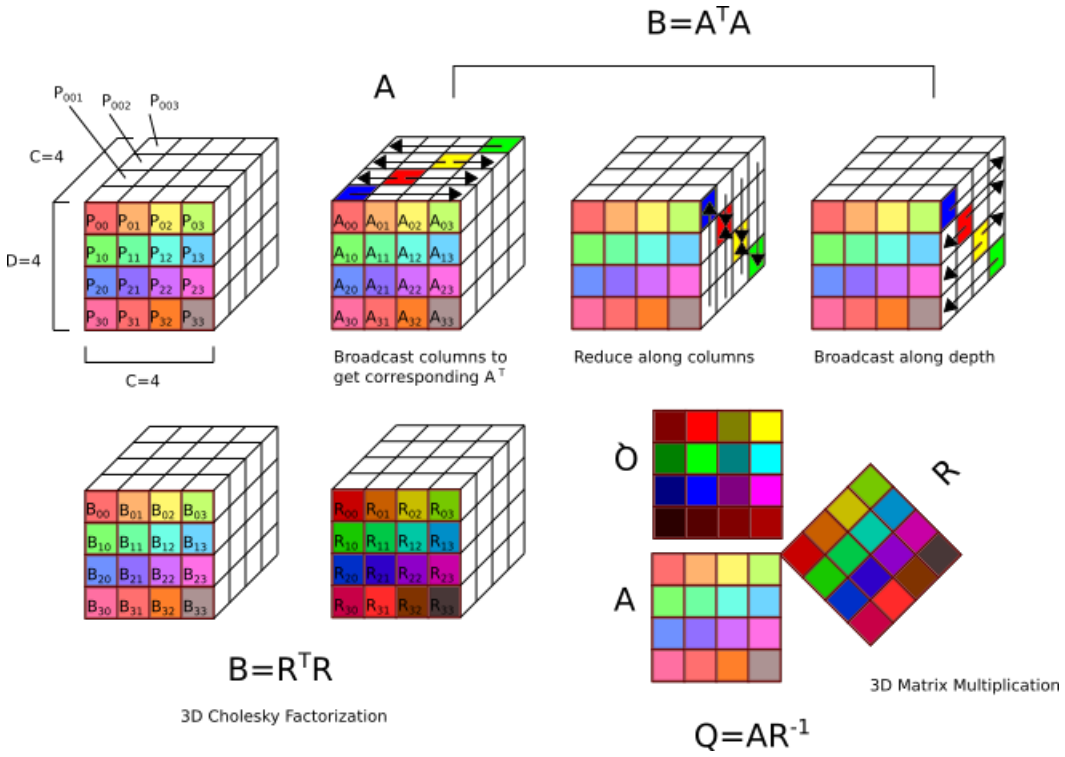

Figure 2.6: Illustration of the steps required in performing CholeskyQR over a 3D processor grid.

thus be generalized, adding additonal processor grid parameters able to be tuned. Tunable algorithms solved over tunable grids can achieve a wide range of asymptotic costs. These costs can be shown to interpolate between known algorithms on specific grids. Reduced communication is only possible when the correct processor grid is utilized. Skinny matrices can not take full advantage of the resources provided by a 3D grid, while square matrices overload the resource capacity that skinny rectangular grids provide. Finding the optimal processor grid for a specific algorithm is nontrivial.

The CholeskyQR2_Tunable algorithm can be seen as a generalization of CholeskyQR2_1D and CholeskyQR2_3D. We define a $c \times d \times c$ rectangular processor grid Π_T that partitions the $m \times n$ matrix A into rectangular blocks of size $\frac{m}{d} \times \frac{n}{c}$. CholeskyQR2_Tunable effectively utilizes its tunable grid by performing $\frac{d}{c}$ simultaneous instances of CholeskyFactorization3D on cubic grid partitions of dimension c. This allows each grid partition to avoid further communication with other partitions because each has the necessary data to compute the final step $Q = AR^{-1}$. In order to allow these simulataneous cholesky factorizations, a few extra steps are needed to get the required data onto the correct processor.

As in CholeskyQR_3D, $\Pi_T[k,j,k]$ broadcasts A_{jk} to $\Pi[:,j,k], \forall j \in [0,d-1], k \in [0,c-1]$. With its received block and local block, each processor can perform matrix multiplication $X_{ji} \leftarrow W_{ji}^T A_{ji}$. In order for c cubic partitions of Π_T to own the same matrix $B = A^T A$, we peform the following two steps. First, we subdivide Π_T along dimension y into $\frac{d}{c}$ contiguous groups of size c. These groups, along with their mirrors along dimension z, will belong to the same cubic partition when calling CholeskyFactorization3D. $\Pi_T[i, c \cdot \lfloor \frac{j}{c} \rfloor : (c+1) \cdot \lfloor \frac{j}{c} \rfloor, k]), \forall i,k \in [0,c-1]$ participates in an AllReduce in order to simulate the horizontal summing of rows in a linear combination. Processor $\Pi_T[i,j,k], \forall i,k \in [0,c-1], \forall j$ such that $j \bmod c = k$ for its corresponding z dimension index k, is the only processor that must retain the communicated data. To get the data from the other $\frac{d}{c} - 1$ groups on each slice, we subdivide Π_T along dimension y into c groups of size $\frac{d}{c}$, where each processor belonging to the same group is a step size c away. The AllReduce is performed on subcommunicator $\Pi_T[i, j : c : d, k], \forall i,k \in [0,c-1], \forall j = k$ on dimension-z index k, resulting in every processor in that subcommunicator owning the correct region of $B = A^T A$. A final broadcast from root $\Pi_T[i,j,k], \forall i,k \in [0,c-1], \forall j$ such that $j \bmod c = k$ for its corresponding z dimension index k, along $\Pi_T[i,j,:], \forall i \in [0,c-1], j \in [0,d-1]$ ensures that B is replicated $\frac{d}{c} \cdot c$ times among subcommunicators of size c^2.

Now that B is distributed as described above, $\frac{d}{c}$ simultaneous instances of CholeskyFactorization3D and MatrixMultiplication3D are performed. CholeskyQR2_Tunable requires calling CholeskyQR_Tunable twice and performing a final MatrixMultiplication3D to solve for R.

It should be clear from both Figure 2.7 and the pseudocode presented in Algorithms 9 and 10 that CholeskyQR2_Tunable tunes pieces of both the 1D and 3D CholeskyQR2 algorithms in order to most efficiently span the grid range $c \in [1, P^{\frac{1}{3}}]$. To prove that this algorithm achieves the same costs as CholeskyQR_1D and CholeskyQR2_3D with grid sizes $c = 1, c = P^{\frac{1}{3}}$, respectively, we provide a cost analysis below.

Algorithm 9 $[Q, R] \leftarrow$ **CholeskyQR_Tunable** $(A, m, n, \Pi_T, i, j, k)$

Require: Π_T has P processors arranged in a tunable grid of size $c \times d \times c$ for any integer c in range $[0, P^{\frac{1}{3}} - 1]$. A is $m \times n$ and is replicated on $\Pi_T[:, :, k], \forall k \in [0, c-1]$. Each processor $\Pi_T[i, j, k]$ owns a (cyclic) blocked partition of A known as A_{ji}. A_{ji} is packed into a 1D array of size $\frac{mn}{dc}$, where it owns a rectangular piece of size $\frac{m}{d} \times \frac{n}{c}$. Let W, X, Y, Z, and R^{-1} be temporary arrays distributed the same as A.

1: **Bcast** $(A_{jk}, W_{ji}, k, \Pi_T[:, j, k])$ ▷ Broadcast from root k across row i
2: $X_{ji} \leftarrow$ **seq-MM** $\left(W_{ji}^T, A_{ji}, \frac{n}{c}, \frac{m}{d}, \frac{n}{c}\right)$
3: **AllReduce** $\left(X_{ji}, Y_{ji}, \Pi_T[i, c \cdot \lfloor \frac{j}{c} \rfloor : (c+1) \cdot \lfloor \frac{j}{c} \rfloor, k]\right)$ ▷ AllReduce among groups of c along each column
4: **AllReduce** $(Y_{ji}, Z_{ji}, \Pi_T[i, c : c : d, k])$ ▷ AllReduce among groups of c of size c distance away along each column
5: **Bcast** $(Z_{ji}, Z_{ji}, k, \Pi_T[i, j, :])$ ▷ Every 2D slice owns the same matrix $B = A^T$
6: Define $\Pi_3 \leftarrow \Pi_T[:, c \cdot \lfloor \frac{j}{c} \rfloor : (c+1) \cdot \lfloor \frac{j}{c} \rfloor, :]$ ▷ Split rectangular processor grid into cubic grid of dimension c
7: $R^T, R^{-T} \leftarrow$ **CholeskyFactorization3D** $\left(Z, n, \frac{n}{P^{\frac{2}{3}}}, \Pi_3, i, j \bmod c, k\right)$
8: $Q \leftarrow$ **MatrixMultiplication3D** $\left(A, R^{-1}, m, n, n, \Pi_3, i, j \bmod c, k\right)$

Ensure: $A = QR$, where Q and R are distributed the same as A. Q is $m \times n$ and R is an upper triangular matrix of dimension n.

Algorithm 10 $[Q, R] \leftarrow$ **CholeskyQR2_Tunable** $(A, m, n, \Pi_T, i, j, k)$

Require: Same requirements as Algorithm 7.

1: $X, Y \leftarrow$ **CholeskyQR_Tunable** $(A, m, n, \Pi_T, i, j, k)$
2: $Q, Z \leftarrow$ **CholeskyQR_Tunable** $(X, m, n, \Pi_T, i, j, k)$
3: Define $\Pi_1 \leftarrow \Pi_T[:, c \cdot \lfloor \frac{j}{c} \rfloor : (c+1) \cdot \lfloor \frac{j}{c} \rfloor, :]$
4: $R \leftarrow$ **MatrixMultiplication3D** $(Z, Y, n, \Pi_1, i, j \bmod c, k)$

Ensure: Same requirements as Algorithm 7.

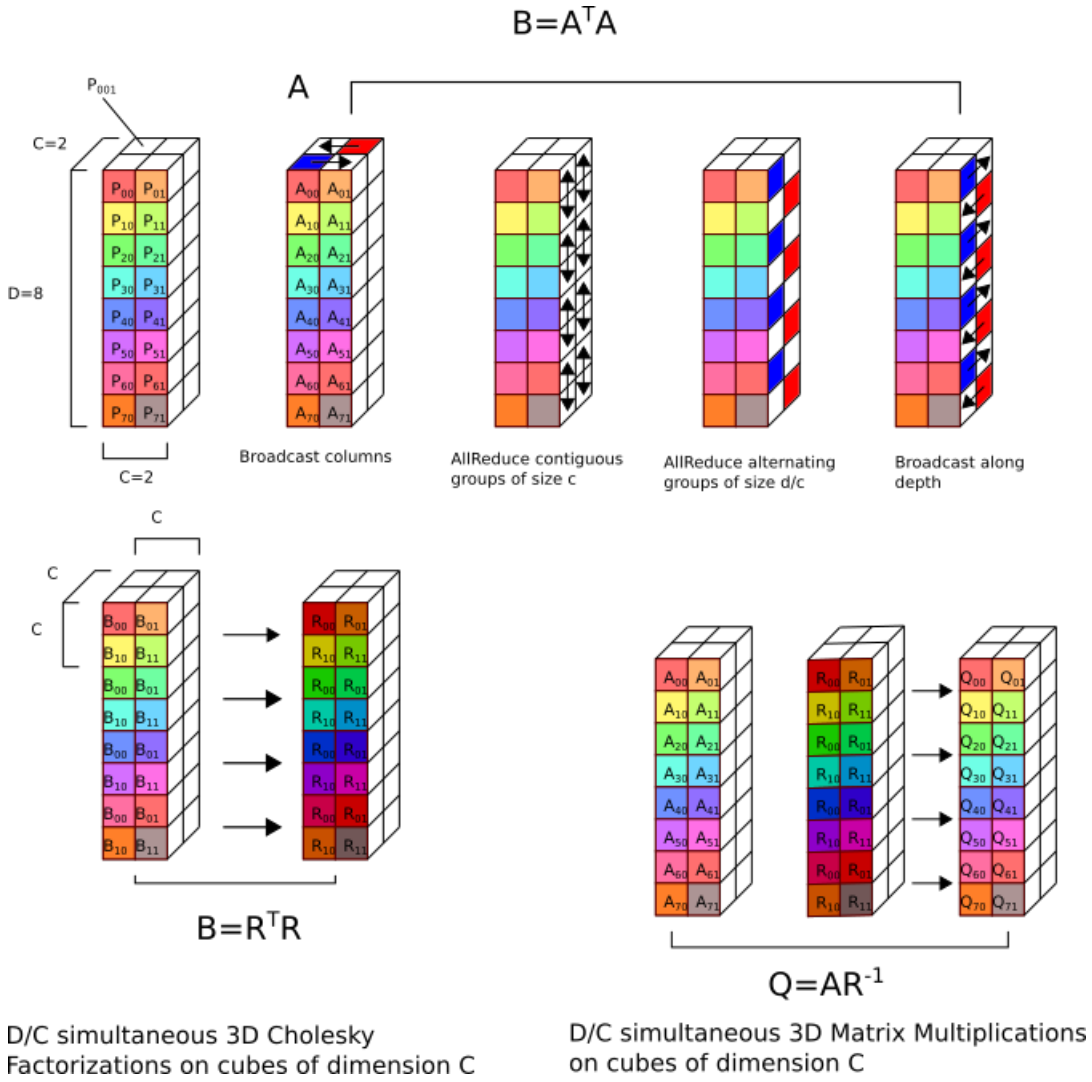

Figure 2.7: Illustration of the steps required to perform CholeskyQR over a tunable processor grid.

Chapter 3

Algorithm Cost Analysis

We present a cost analysis for the following algorithms: MatrixMultiplication3D, Cholesky-Factorization3D, CholeskyQR2_1D, CholeskyQR2_3D, and CholeskyQR2_Tunable. We analyze the effects differences in matrix sizes and processor grid dimensions have on their asymptotic costs. Finally, we compare the cost of the tunable Cholesky-QR2 algorithm against existing QR factorization algorithms.

3.1 Preliminaries

To analyze these costs, we use a simple α-β model as defined below.

$$\begin{aligned} &\alpha \rightarrow \text{cost of sending or receiving a single message} \\ &\beta \rightarrow \text{cost of moving a single word of data among processors} \\ &\gamma \rightarrow \text{cost of computing a single floating point operation} \end{aligned} \tag{3.1}$$

We define a few sequential routines used in Chapter 2 and give their asymptotic costs.

$$\begin{aligned} T^{\alpha-\beta}_{\text{seq-MM}}(m,n,k) &= \mathcal{O}(mnk)\cdot\gamma \\ T^{\alpha-\beta}_{\text{seq-Subtract}}(m,n) &= \mathcal{O}(mn)\cdot\gamma \\ T^{\alpha-\beta}_{\text{seq-Syrk}}(m,n) &= \mathcal{O}\left(mn^2\right)\cdot\gamma \\ T^{\alpha-\beta}_{\text{seq-Cholesky}}(n) &= \mathcal{O}\left(n^3\right)\cdot\gamma \\ T^{\alpha-\beta}_{\text{seq-TriInv}}(n) &= \mathcal{O}\left(n^3\right)\cdot\gamma \end{aligned} \tag{3.2}$$

We also want to define a unit-step function as follows: $\delta(x) = \begin{cases} 0 & x \leq 1 \\ 1 & x > 1 \end{cases}$

3.2 Collective Communication

The costs of the following collective routines can be obtained by a butterfly schedule, where n is the number of bytes of data being moved and P is the number of processors involved in the communication.

$$
\begin{aligned}
T^{\alpha-\beta}_{\text{Bcast}}(n,P) &= 2\log_2 P\cdot\alpha + 2n\delta(P)\cdot\beta \\
T^{\alpha-\beta}_{\text{AllReduce}}(n,P) &= 2\log_2 P\cdot\alpha + 2n\delta(P)\cdot\beta + n\delta(P)\cdot\gamma \\
T^{\alpha-\beta}_{\text{AllGather}}(n,P) &= \log_2 P\cdot\alpha + n\delta(P)\cdot\beta
\end{aligned}
$$

3.3 Matrix Multiplication

We analyze the cost of MatrixMultiplication3D for multiplying an $m\times n$ matrix by an $n\times k$ matrix over a 3D processor grid in Table 3.1.

Table 3.1: Costs of MatrixMultiplication3D.

Line Number	Cost
1	$2\log_2 P^{\frac{1}{3}}\cdot\alpha + \frac{2mn\delta(P)}{P^{\frac{2}{3}}}\cdot\beta$
2	$2\log_2 P^{\frac{1}{3}}\cdot\alpha + \frac{2nk\delta(P)}{P^{\frac{2}{3}}}\cdot\beta$
3	$\mathcal{O}\left(\frac{mnk}{P}\right)\cdot\gamma$
4	$2\log_2 P^{\frac{1}{3}}\cdot\alpha + \frac{2mk\delta(P)}{P^{\frac{2}{3}}}\cdot\beta + \frac{mk\delta(P)}{P^{\frac{2}{3}}}\cdot\gamma$

$$
\begin{aligned}
T^{\alpha-\beta}_{\text{MatrixMultiplication3D}}(m,n,k,P) &= 6\log_2 P^{\frac{1}{3}}\cdot\alpha + \frac{(2mn+2nk+2mk)\,\delta(P)}{P^{\frac{2}{3}}}\cdot\beta \\
&\quad + \mathcal{O}\left(\frac{mk\delta(P)}{P^{\frac{2}{3}}} + \frac{mnk}{P}\right)\cdot\gamma \\
&= \mathcal{O}\left(\log P\cdot\alpha + \frac{(mn+nk+mk)\,\delta(P)}{P^{\frac{2}{3}}}\cdot\beta + \frac{mnk}{P}\cdot\gamma\right)
\end{aligned}
$$

3.4 Cholesky Factorization

The cost of CholeskyFactorization3D is determined by the cost of matrix multiplication at each recursive level and the total cost of the base case. In order to utilize the entire 3D processor grid, Π, for each instance of MatrixMultiplication3D, we recurse into smaller

matrix windows without splitting Π. This influences our decision to use a cyclic distribution because each processor owns an active part of the shrinking matrix window. Upon reaching matrix window size $\frac{n}{2^z} = n_o$, matrices L and L^{-1} are solved explicitely using seq-Cholesky and seq-TriInv. An allgather routine is needed to load the data scattered across the matrix window into $\Pi[:,:,k], \forall k \in [0, P^{\frac{1}{3}} - 1]$. The combined communication cost of the base case must not dominate the cost of the MatrixMultiplication3D.

We analyze the cost of CholeskyFactorization3D in Table 3.2.

Table 3.2: Costs of CholeskyFactorization3D.

Line Number	Cost
2	$\log_2 P^{\frac{2}{3}} \cdot \alpha + n_o^2 \delta(P) \cdot \beta$
3	$\mathcal{O}(n_o^3) \cdot \gamma$
4	$\mathcal{O}(n_o^3) \cdot \gamma$
7	$\delta(P) \cdot \alpha + \frac{\frac{n}{2}\left(\frac{n}{2}+1\right)\delta(P)}{2P^{\frac{2}{3}}} \cdot \beta$
8	$2\log_2 P \cdot \alpha + \frac{\left(5n^2+2n\right)\delta(P)}{4P^{\frac{2}{3}}} \cdot \beta + \mathcal{O}\left(\frac{n^3}{P}\right) \cdot \gamma$
9	$\delta(P) \cdot \alpha + \frac{n^2\delta(P)}{4P^{\frac{2}{3}}} \cdot \beta$
10	$2\log_2 P \cdot \alpha + \frac{3n^2\delta(P)}{2P^{\frac{2}{3}}} \cdot \beta + \mathcal{O}\left(\frac{n^3}{P}\right) \cdot \gamma$
11	$\mathcal{O}\left(\frac{n^2}{P^{\frac{2}{3}}}\right) \cdot \gamma$
13	$2\log_2 P \cdot \alpha + \frac{\left(5n^2+2n\right)\delta(P)}{4P^{\frac{2}{3}}} \cdot \beta + \mathcal{O}\left(\frac{n^3}{P}\right) \cdot \gamma$
14	$\mathcal{O}\left(\frac{n^2}{P^{\frac{2}{3}}}\right) \cdot \gamma$
15	$2\log_2 P \cdot \alpha + \frac{\left(5n^2+2n\right)\delta(P)}{4P^{\frac{2}{3}}} \cdot \beta + \mathcal{O}\left(\frac{n^3}{P}\right) \cdot \gamma$

$$\begin{aligned} T^{\alpha-\beta}_{\text{CholeskyBaseCase}}(n_o, P) &= \log_2 P^{\frac{2}{3}} \cdot \alpha + n_o^2 \delta(P) \cdot \beta + \mathcal{O}\left(n_o^3\right) \cdot \gamma \\ &= \mathcal{O}\big(\log P \cdot \alpha + n_o^2 \delta(P) \cdot \beta + n_o^3 \cdot \gamma\big) \end{aligned}$$

Choice of n_o depends on the non-recursive communication cost. Because $\frac{n}{2^z} = n_o$, our algorithm must compute $\frac{n}{n_o}$ AllGathers.

$$\begin{aligned}
T^{\alpha-\beta}_{\text{CholeskyFactorization3D}}(n,P) &= 2T^{\alpha-\beta}_{\text{CholeskyFactorization3D}}\left(\frac{n}{2},P\right) + \left(8\log_2 P + 2\delta(P)\right)\cdot\alpha \\
&\quad + \frac{\left(22.5n^2+7n\right)\delta(P)}{4P^{\frac{2}{3}}}\cdot\beta + \mathcal{O}\left(\frac{n^3}{P}\right)\cdot\gamma \\
&= 2T^{\alpha-\beta}_{\text{CholeskyFactorization3D}}\left(\frac{n}{2},P\right) + \mathcal{O}\left(\log P\cdot\alpha + \frac{n^2\delta(P)}{P^{\frac{2}{3}}}\cdot\beta + \frac{n^3}{P}\cdot\gamma\right) \\
&= 2^z T^{\alpha-\beta}_{\text{CholeskyBaseCase}}(n_o,P) + \sum_{q=0}^{z-1} 2^q\cdot\mathcal{O}\left(\log P\cdot\alpha + \frac{\left(\frac{n}{2^q}\right)^2\delta(P)}{P^{\frac{2}{3}}}\cdot\beta + \frac{\left(\frac{n}{2^q}\right)^3}{P}\cdot\gamma\right) \\
&= \mathcal{O}\left(\frac{n\log P}{n_o}\cdot\alpha + nn_o\delta(P)\cdot\beta + nn_o^2\cdot\gamma\right) \\
&\quad + \mathcal{O}\left(\frac{n\log P}{n_o}\cdot\alpha + \frac{n^2\delta(P)}{P^{\frac{2}{3}}}\cdot\beta + \frac{n^3}{P}\cdot\gamma\right)
\end{aligned}$$

Choice of $\frac{n}{n_o}$ creates a tradeoff between the synchronization cost and the communication cost. We elect to match the communication cost at the expense of an increase in synchronization, giving the relation $n_o = \frac{n}{P^{\frac{2}{3}}}$. The final cost of the 3D algorithm is the following:

$$T_{\text{CholeskyFactorization}}(n,P) = \mathcal{O}\left(P^{\frac{2}{3}}\log P\cdot\alpha + \frac{n^2\delta(P)}{P^{\frac{2}{3}}}\cdot\beta + \frac{n^3}{P}\cdot\gamma\right)$$

3.5 QR Factorization

We provide a brief analysis of the existing 1D CholeskyQR2 algorithm and a thorough analysis of our new 3D and tunable CholeskyQR2 algorithms. We show that with the correct parameters, the tunable algorithm achieves the same communication cost as the 1D and 3D algorithms. Finally, we derive the optimal communication cost achieved by the tunable algorithm.

3.5.1 1D Algorithm

See Table 3.3 for the costs attained in the CholeskyQR_1D algorithm. Note that the AllReduce in line 2 need only send the upper-triangle of symmetric matrix B_p, so a factor of $\frac{1}{2}$ is applied to the horizontal bandwidth cost.

Table 3.3: Costs of CholeskyQR_1D.

Line Number	Cost
1	$\mathcal{O}\left(\frac{n^2m}{P}\right)\cdot\gamma$
2	$2\log_2 P\cdot\alpha+n^2\delta(P)\cdot\beta+\frac{n^2}{2}\cdot\gamma$
3	$\mathcal{O}(n^3)\cdot\gamma$
4	$\mathcal{O}(n^3)\cdot\gamma$
5	$\mathcal{O}\left(\frac{n^2m}{P}\right)\cdot\gamma$

$$T^{\alpha-\beta}_{\text{CholeskyQR\_1D}}(m,n,P)=2\log_2 P\cdot\alpha+n^2\delta(P)\cdot\beta+\mathcal{O}\left(\frac{n^2m}{P}+n^3\right)\cdot\gamma \tag{3.3}$$

The costs in Table 3.4 are attained in the CholeskyQR2_1D algorithm.

Table 3.4: Costs of CholeskyQR2_1D.

Line Number	Cost
1	$2\log_2 P\cdot\alpha+n^2\delta(P)\cdot\beta+\mathcal{O}\left(\frac{n^2m}{P}+n^3\right)\cdot\gamma$
2	$2\log_2 P\cdot\alpha+n^2\delta(P)\cdot\beta+\mathcal{O}\left(\frac{n^2m}{P}+n^3\right)\cdot\gamma$
3	$\mathcal{O}(n^3)\cdot\gamma$

$$\begin{aligned}T^{\alpha-\beta}_{\text{CholeskyQR2\_1D}}(m,n,P)&=4\log_2 P\cdot\alpha+2n^2\delta(P)\cdot\beta+\mathcal{O}\left(\frac{n^2m}{P}+n^3\right)\cdot\gamma\\&=\mathcal{O}\left(\log P\cdot\alpha+n^2\delta(P)\cdot\beta+\left(\frac{n^2m}{P}+n^3\right)\cdot\gamma\right)\end{aligned} \tag{3.4}$$

Varying $\frac{m}{P}$ and n can lead to different asymptotic costs and advantages and disadvantages in practice. See Table 3.5.

Table 3.5: Costs of CholeskyQR2_1D with varying block sizes.

$\frac{m}{P}>n$	$\mathcal{O}\left(\log P\cdot\alpha+n^2\delta(P)\cdot\beta+\frac{n^2m}{P}\cdot\gamma\right)$
$\frac{m}{P}\leq n$	$\mathcal{O}(\log P\cdot\alpha+n^2\delta(P)\cdot\beta+n^3\cdot\gamma)$

This algorithm achieves poor scalability in communication, computation, and memory footprint. Regardless of P, the AllGather distributes an $n\times n$ matrix onto each processor and as n grows the matrix won't fit into a reasonably sized memory. Results from [6] show that this algorithm performs well when $m\gg n$. Therefore, this algorithm can only be used

when $m \gg n$. Below, we show that CholeskyQR2_Tunable can scale efficiently on a tunable processor grid.

3.5.2 3D Algorithm

The costs in Table 3.6 are attained in the CholeskyQR_3D algorithm.

Table 3.6: Costs of CholeskyQR_3D.

Line Number	Cost
1	$2\log_2 P^{\frac{1}{3}} \cdot \alpha + \frac{2nm\delta(P)}{P^{\frac{2}{3}}} \cdot \beta$
2	$\mathcal{O}\left(\frac{n^2m}{P}\right) \cdot \gamma$
3	$2\log_2 P^{\frac{1}{3}} \cdot \alpha + \frac{2n^2\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^2\delta(P)}{P^{\frac{2}{3}}} \cdot \gamma$
4	$2\log_2 P^{\frac{1}{3}} \cdot \alpha + \frac{2n^2\delta(P)}{P^{\frac{2}{3}}} \cdot \beta$
5	$\mathcal{O}\left(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{n^2\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^3}{P} \cdot \gamma\right)$
6	$2\log_2 P \cdot \alpha + \frac{\left(4mn+n^2+nP^{\frac{1}{3}}\right)\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \mathcal{O}\left(\frac{n^2m}{P}\right) \cdot \gamma$

$$T^{\alpha-\beta}_{\text{CholeskyQR\_3D}}(m,n,P) = \mathcal{O}\left(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{(n^2+nm)\,\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^2m+n^3}{P} \cdot \gamma\right) \quad (3.5)$$

The costs in Table 3.7 are attained in the CholeskyQR2_3D algorithm.

Table 3.7: Costs of CholeskyQR2_3D.

Line Number	Cost
1	$\mathcal{O}\left(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{\left(n^2+nm\right)\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^2m+n^3}{P} \cdot \gamma\right)$
2	$\mathcal{O}\left(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{\left(n^2+nm\right)\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^2m+n^3}{P} \cdot \gamma\right)$
3	$2\log_2 P \cdot \alpha + \frac{\left(3n^2+6nP^{\frac{1}{3}}\right)\delta(P)}{2P^{\frac{2}{3}}} \cdot \beta + \mathcal{O}\left(\frac{n^3}{P}\right) \cdot \gamma$

$$T^{\alpha-\beta}_{\text{CholeskyQR2\_3D}}(m,n,P) = \mathcal{O}\left(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{(n^2+nm)\,\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^2m+n^3}{P} \cdot \gamma\right) \quad (3.6)$$

Table 3.8: Costs of CholeskyQR2_3D with varying block sizes.

$m > n$	$\mathcal{O}\left(P^{\frac{2}{3}}\log P\cdot\alpha+\frac{nm\delta(P)}{P^{\frac{2}{3}}}\cdot\beta+\frac{n^2m}{P}\cdot\gamma\right)$
$m \leq n$	$\mathcal{O}\left(P^{\frac{2}{3}}\log P\cdot\alpha+\frac{n^2\delta(P)}{P^{\frac{2}{3}}}\cdot\beta+\frac{n^3}{P}\cdot\gamma\right)$

This algorithm is most communication efficient when $m = n$. See Table 3.8 for the costs attained by CholeskyQR2_3D when varying block sizes.

3.5.3 Tunable Algorithm

The CholeskyQR_Tunable algorithm attains the costs in Table 3.9.

Table 3.9: Costs of CholeskyQR_Tunable.

Line Number	Cost
1	$2\log_2 c\cdot\alpha+\frac{2mn\delta(c)}{dc}\cdot\beta$
2	$\mathcal{O}\left(\frac{n^2m}{c^2d}\right)$
3	$2\log_2 c\cdot\alpha+\frac{2n^2\delta(c)}{c^2}\cdot\beta+\frac{n^2\delta(c)}{c^2}\cdot\gamma$
4	$2\log_2 \frac{d}{c}\cdot\alpha+\frac{2n^2\delta(d)}{c^2}\cdot\beta+\frac{n^2\delta(d)}{c^2}\cdot\gamma$
5	$2\log_2 c\cdot\alpha+\frac{2mn\delta(c)}{dc}\cdot\beta$
6	$\mathcal{O}\left(c^2\log c^3\cdot\alpha+\frac{n^2\delta(c)}{c^2}\cdot\beta+\frac{n^3}{c^3}\cdot\gamma\right)$
7	$2\log_2 c^3\cdot\alpha+\left(\frac{4mn\delta(c)}{dc}+\frac{\left(n^2+nc\right)\delta(c)}{c^2}\right)\cdot\beta+\mathcal{O}\left(\frac{n^2m}{c^2d}\right)\cdot\gamma$

$$\begin{aligned}T^{\alpha-\beta}_{\text{CholeskyQR\_Tunable}}(m,n,c,d) &= \mathcal{O}\Bigg(\left(c^2\log c+\log\frac{d}{c}\right)\cdot\alpha\\ &\quad+\left(\frac{mn\delta(c)}{dc}+\frac{n^2\delta(c)}{c^2}+\frac{n^2\delta(d)}{c^2}\right)\cdot\beta+\left(\frac{n^3}{c^3}+\frac{n^2m}{c^2d}\right)\cdot\gamma\Bigg)\\ &= \mathcal{O}\Bigg(c^2\log P\cdot\alpha+\frac{cmn\delta(c)+n^2d\delta(P)}{dc^2}\cdot\beta\\ &\quad+\frac{n^3d+n^2mc}{c^3d}\cdot\gamma\Bigg)\end{aligned}\tag{3.7}$$

See Table 3.10 for the costs attained in the CholeskyQR2_Tunable algorithm.

Table 3.10: Costs of CholeskyQR2_Tunable.

Line Number	Cost
1	$\mathcal{O}\left(c^2\log P\cdot\alpha+\frac{cmn\delta(c)+n^2d\delta(P)}{dc^2}\cdot\beta+\frac{n^3d+n^2mc}{c^3d}\cdot\gamma\right)$
2	$\mathcal{O}\left(c^2\log P\cdot\alpha+\frac{cmn\delta(c)+n^2d\delta(P)}{dc^2}\cdot\beta+\frac{n^3d+n^2mc}{c^3d}\cdot\gamma\right)$
3	$2\log_2 c^3\cdot\alpha+\frac{\left(3n^2+6nc\right)\delta(c)}{2c^2}\cdot\beta+\mathcal{O}\left(\frac{n^3}{c^3}\right)\cdot\gamma$

$$T^{\alpha-\beta}_{\text{CholeskyQR2\_Tunable}}(m,n,c,d) = \mathcal{O}\Bigg(c^2 \log P \cdot \alpha + \frac{cmn\delta(c) + n^2 d\delta(P)}{dc^2} \cdot \beta + \frac{n^3 d + n^2 mc}{c^3 d} \cdot \gamma\Bigg) \tag{3.8}$$

We can show that costs attained by CholeskyQR2_Tunable correctly interpolates between the costs of CholeskyQR2_1D and CholeskyQR2_3D. Note that our $c \times d \times c$ grid requires $P = c^2 d$ and $d \geq c$.

$$\begin{aligned} T^{\alpha-\beta}_{\text{CholeskyQR2\_Tunable}}(m,n,1,P) &= \mathcal{O}\Bigg(1^2 \log P \cdot \alpha + \frac{mn\delta(1) + n^2 P\delta(P)}{P \cdot 1^2} \cdot \beta + \frac{n^3 P + n^2 m}{1^3 \cdot P} \cdot \gamma\Bigg) \\ &= \mathcal{O}\left(\log P \cdot \alpha + n^2\delta(P) \cdot \beta + \left(n^3 + \frac{n^2 m}{P}\right) \cdot \gamma\right) \end{aligned} \tag{3.9}$$

$$\begin{aligned} T^{\alpha-\beta}_{\text{CholeskyQR2\_Tunable}}\left(m,n,P^{\frac{1}{3}},P^{\frac{1}{3}}\right) &= \mathcal{O}\Bigg(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{mnP^{\frac{1}{3}}\delta\left(P^{\frac{1}{3}}\right) + n^2 P^{\frac{1}{3}}\delta(P)}{P} \cdot \beta + \frac{n^3 P^{\frac{1}{3}} + n^2 m P^{\frac{1}{3}}}{P^{\frac{4}{3}}} \cdot \gamma\Bigg) \\ &= \mathcal{O}\left(P^{\frac{2}{3}} \log P \cdot \alpha + \frac{(n^2 + nm)\,\delta(P)}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^3 + n^2 m}{P} \cdot \gamma\right) \end{aligned} \tag{3.10}$$

The advantage of using a tunable grid lies in the ability to frame the shape of the grid around the shape of rectangular $m \times n$ matrix A. If $m \gg n$, it might make sense to let $c = 1$, giving a 1D grid of shape $1 \times P \times 1$ on which we can run CholeskyQR2_1D. If $m = n$, it might be most performant to allow $c = P^{\frac{1}{3}}$, producing a cubic grid of dimensions $P^{\frac{1}{3}}$. If A is between these shapes, our tunable grid can factor $A = QR$ at asymptotically less cost. Optimal communication can be attained by ensuring that the grid perfectly fits the dimensions of A, or that the dimensions of the grid are proportional to the dimensions of the matrix. We derive the cost for the optimal ratio $\frac{m}{d} = \frac{n}{c}$ below. For clarity, we assume all δ-terms are 1.

Using equation $P = c^2 d$ and $\frac{m}{d} = \frac{n}{c}$, solve for d, c in terms of m, n, P. Solving the system of equations yields $c = \left(\frac{Pn}{m}\right)^{\frac{1}{3}}, d = \left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}$. We can plug these values into the cost of

CholeskyQR2_Tunable to find the optimal cost.

$$
\begin{aligned}
T^{\alpha-\beta}_{\text{CholeskyQR2\_Tunable}}\left(m,n,\left(\frac{Pn}{m}\right)^{\frac{1}{3}},\left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}\right) &= \mathcal{O}\Bigg(\left(\frac{Pn}{m}\right)^{\frac{2}{3}}\log P\cdot\alpha \\
&+\frac{\left(\frac{Pn}{m}\right)^{\frac{1}{3}}mn+n^2\left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}}{\left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}\left(\frac{Pn}{m}\right)^{\frac{2}{3}}}\cdot\beta+\frac{n^3\left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}+n^2m\left(\frac{Pn}{m}\right)^{\frac{1}{3}}}{\left(\frac{Pn}{m}\right)\left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}}\cdot\gamma\Bigg) \\
&= \mathcal{O}\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}}\log P\cdot\alpha+\left(\frac{n^2m}{P}\right)^{\frac{2}{3}}\cdot\beta+\frac{n^2m}{P}\cdot\gamma\right)
\end{aligned}
\tag{3.11}
$$

See Table 3.11 for an overview on the costs attained by each variant.

Table 3.11: Overview of costs attained by CholeskyQR2 algorithm variants.

Grid shape	Metric	Cost
$1\times P\times 1$	# of messages	$\mathcal{O}\left(\log P\right)$
	# of words	$\mathcal{O}\left(n^2\delta\left(P\right)\right)$
	# of flops	$\mathcal{O}\left(\frac{n^2m}{P}+n^3\right)$
	Memory footprint	$\mathcal{O}\left(\frac{mn}{P}+n^2\right)$
$P^{\frac{1}{3}}\times P^{\frac{1}{3}}\times P^{\frac{1}{3}}$	# of messages	$\mathcal{O}\left(P^{\frac{2}{3}}\log P\right)$
	# of words	$\mathcal{O}\left(\frac{\left(n^2+nm\right)\delta(P)}{P^{\frac{2}{3}}}\right)$
	# of flops	$\mathcal{O}\left(\frac{n^2m+n^3}{P}\right)$
	Memory footprint	$\mathcal{O}\left(\frac{mn+n^2}{P^{\frac{2}{3}}}\right)$
$c\times d\times c$	# of messages	$\mathcal{O}\left(c^2\log P\right)$
	# of words	$\mathcal{O}\left(\frac{cmn\delta(c)+n^2d\delta(P)}{dc^2}\right)$
	# of flops	$\mathcal{O}\left(\frac{n^3d+n^2mc}{c^3d}\right)$
	Memory footprint	$\mathcal{O}\left(\frac{mnc+n^2d}{c^2d}\right)$
optimal	# of messages	$\mathcal{O}\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}}\log P\right)$
	# of words	$\mathcal{O}\left(\left(\frac{n^2m}{P}\right)^{\frac{2}{3}}\delta\left(P\right)\right)$
	# of flops	$\mathcal{O}\left(\frac{n^2m}{P}\right)$
	Memory footprint	$\mathcal{O}\left(\left(\frac{n^2m}{P}\right)^{\frac{2}{3}}\right)$

Chapter 4

Conclusion

4.1 Conclusion

In this work, we have developed algorithms that efficiently extend CholeskyQR2 to matrices with an arbitrary number of rows and columns. Through the use of a tunable processor grid of size $c \times d \times c$, CholeskyQR2 has been generalized to a tunable algorithm best equipped to exploit the shape of A. Our algorithm, CholeskyQR2_Tunable, capitalizes on the flexibility of its tunable grid to become the first practical QR factorization algorithm to achieve optimally minimal communication within logarithmic latency factors for rectangular matrices.

4.2 Future Work

We currently have a draft implementation of the CholeskyQR2_Tunable algorithm. The next step in this line of work should be to evaluate and tune this proposed algorithm on a large-scale distributed-memory architecture. Further avenues to explore include adding a study of vertical communication cost to each of the algorithms, studying the stability of CholeskyQR_3D and CholeskyQR_Tunable to identify any algorithmic changes that can improve upon the stability of CholeskyQR, and tuning local block sizes to find the most performant size. Others include developing a recursive tunable Cholesky factorization algorithm that can be tuned to fit machine memory sizes efficiently, as well as developing new algorithms for tunable processor grids that provably reduce communication.

Bibliography

[1] S. Graham, M. Snir, C. Patterson, and N. R. C. U. C. on the Future of Supercomputing, Getting up to speed: the future of supercomputing. National Academies Press, 2005. [Online]. Available: https://books.google.com/books?id=8KlQAAAAMAAJ

[2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, ScaLAPACK User's Guide, J. J. Dongarra, Ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.

[3] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms," in Euro-Par 2011 Parallel Processing, ser. Lecture Notes in Computer Science, E. Jeannot, R. Namyst, and J. Roman, Eds. Springer Berlin Heidelberg, 2011, vol. 6853, pp. 90–109. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23397-5_10

[4] E. Solomonik, A. Buluc, and J. Demmel, "Minimizing communication in all-pairs shortest paths," in Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013, pp. 548–559.

[5] A. Tiskin, "Bulk-synchronous parallel Gaussian elimination," Journal of Mathematical Sciences, vol. 108, no. 6, pp. 977–991, 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1013588221172

[6] T. Fukaya, Y. Nakatsukasa, Y. Yanagisawa, and Y. Yamamoto, "CholeskyQR2: A simple and communication-avoiding algorithm for computing a tall-skinny QR factorization on a large-scale parallel system," in Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ser. ScalA '14. Piscataway, NJ, USA: IEEE Press, 2014. [Online]. Available: http://dx.doi.org/10.1109/ScalA.2014.11 pp. 31–38.

[7] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H. D. Nguyen, and E. Solomonik, "Reconstructing Householder vectors from tall-skinny QR," in Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, ser. IPDPS '14. Washington, DC, USA: IEEE Computer Society, 2014. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2014.120 pp. 1159–1170.

[8] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya, "Roundoff error analysis of the CholeskyQR2 algorithm," Electronic Transactions on Numerical Analysis, vol. 44, pp. 306–326, 2015.

[9] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," SIAM J. Sci. Comput., vol. 34, no. 1, pp. 206–239, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1137/080731992

[10] A. Tiskin, "Communication-efficient parallel generic pairwise elimination," Future Generation Computer Systems, vol. 23, no. 2, pp. 179–188, 2007.

[11] E. Solomonik, G. Ballard, J. Demmel, and T. Hoefler, "A communication-avoiding parallel algorithm for the symmetric eigenvalue problem," ArXiv e-prints, Apr. 2016.

[12] W. F. MColl and A. Tiskin, "Memory-efficient matrix multiplication in the BSP model," Algorithmica, vol. 24, pp. 287–297, 1999.

[13] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," SIAM Journal on Computing, vol. 10, no. 4, pp. 657–675, 1981.

[14] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication," IBM Journal of Research and Development, vol. 39, pp. 575–582, September 1995.

[15] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," Theoretical Computer Science, vol. 71, no. 1, pp. 3 – 28, 1990.

[16] J. Berntsen, "Communication efficient matrix multiplication on hypercubes," Parallel Computing, vol. 12, no. 3, pp. 335–342, 1989.

[17] S. L. Johnsson, "Minimizing the communication time for matrix multiplication on multiprocessors," Parallel Computing, vol. 19, pp. 1235–1257, November 1993.