Formalized Parallel Dense Linear Algebra and its Application to the Generalized Eigenvalue Problem

by

Jack Poulson, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2009

Formalized Parallel Dense Linear Algebra and its Application to the Generalized Eigenvalue Problem

APPROVED BY

SUPERVISING COMMITTEE:

Jeffrey K. Bennighof, Supervisor

Robert A. van de Geijn

To my family.

Acknowledgments

I am indebted to my advisor for not only suggesting that I consider graduate study, but also supporting me through a masters. Without his guidance, I would almost certainly not be where I am today.

I would also like to thank all of the members of our research group, particularly Mark Muller, Jeremiah Palmer, and Josh Haben. Mark's help and input over the past few years has proved invaluable to my research, and thanks to Jeremiah and Josh, our office has always been an amazing place to work.

Lastly, I owe a huge thank you to Robert van de Geijn. He has proven to be an endless source of advice, and I owe the scalable reduction algorithm directly to interactions with him.

Formalized Parallel Dense Linear Algebra and its Application to the Generalized Eigenvalue Problem

Jack Poulson, M.S.E.
The University of Texas at Austin, 2009

Supervisor: Jeffrey K. Bennighof

This thesis demonstrates an efficient parallel method of solving the generalized eigenvalue problem, $K\Phi = M\Phi\Lambda$, where K is symmetric and M is symmetric positive-definite, by first converting it to a standard eigenvalue problem, solving the standard eigenvalue problem, and back-transforming the results. An abstraction for parallel dense linear algebra is introduced along with a new algorithm for forming $A := U^{-T}KU^{-1}$, where U is the Cholesky factor of M, that is up to twice as fast as the ScaLAPACK implementation. Additionally, large improvements over the PBLAS implementations of general matrix-matrix multiplication and triangular solves with many right-hand sides are shown. Significant performance gains are also demonstrated for Cholesky factorizations, and a case is made for using 2D-cyclic distributions with a distribution blocksize of one.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The generalized **e**igen**v**alue **p**roblem (EVP) is the search for nontrivial solutions to

$$Ax = \lambda Bx, \tag{1.1}$$

where we are given $(A, B) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n}$ and are looking for the eigenpair $(\lambda, x) \in \mathbb{R} \times \mathbb{R}^n$. λ is called an eigenvalue of the system, and x is referred to as an eigenvector[20]. If one wishes to find multiple eigenpairs, the problem is often written as

$$AX = BX\Lambda, \tag{1.2}$$

where X is the matrix containing the set of eigenvectors as its columns, and Λ is a diagonal matrix containing the corresponding set of eigenvalues. In the special case where $B = I$, the generalized EVP reduces to the standard eigenvalue problem,

$$AX = X\Lambda. \tag{1.3}$$

The motivation of this thesis is to devise a method for efficiently solving Eq. (1.2) on distributed memory architectures. In particular, we will be looking at the case where both A and B are symmetric, and B is positive-definite. This is true when the matrices are obtained from finite element discretizations of

stiffness and mass operators, when a consistent mass formulation is used. Thus we will instead express Eq. (1.2) as

$$K\Phi = M\Phi\Lambda, \tag{1.4}$$

where K and M are respectively our stiffness and mass matrices. As mentioned, K is assumed to be symmetric, and M is assumed symmetric positive-definite (SPD).

1.1 Background

The current de facto standard for parallel dense linear algebra, ScaLAPACK (Scalable LAPACK)[2, 5], makes use of block 2D-cyclic matrix distributions and a custom communication library called the Basic Linear Algebra Communication Subprograms (BLACS). A different approach to parallel linear algebra was taken by PLAPACK (Parallel Linear Algebra Package)[1, 24], which focused on programmability and derived its communication insights from Physically-Based Matrix Distributions (PBMDs)[11] in terms of the Message Passing Interface (MPI)[14]. However, PLAPACK was implemented under a constraint that prevents its algorithms from being scalable. Generalizing its approach to alleviate this issue complicates matters and is the focus of Chapter Two.

In order to solve the standard form eigenvalue problem in parallel, Hendrickson, Jessup, and Smith[16] suggest a parallelization of the standard three-step process:

1. Tridiagonalize A with Householder reflectors[12] such that $T = Q^T AQ$.

2. Solve the tridiagonal EVP, $TS = S\Lambda$, with the $\mathcal{O}(n^2)$ Algorithm MR^3[8–10].

3. Back-transform the eigenvectors using WY transforms[12, 19].

Hendrickson et al. chose to use a torus-wrap distribution[17] rather than the block torus wrapping used by ScaLAPACK and PLAPACK. They discuss strengths and weaknesses of the elemental distribution and give a full description of parallel algorithms for Householder tridiagonalization and eigenvector back-transformation. While they do mention that Algorithm MR^3 should be trivially parallelizable, this issue is discussed in detail by Bientinesi, Dhillon, and van de Geijn, who refer to their parallel method as PMR^3[3]. Though the HJS method of tridiagonalization was designed solely for perfect-square numbers of processes, our framework will allow it to be readily modified to work for arbitrary numbers of processes. The details of these modifications will be discussed further in Jeremiah Palmer's dissertation. Also, we choose to refer to the torus-wrap mapping as an *elemental* distribution from now on.

The reader will also benefit from becoming familiar with the FLAME project[4, 25]. Van de Geijn et al. have presented a stylistic and systematic methodology for deriving and expressing algorithms that is used heavily in this thesis. In particular, the concept of a loop invariant is crucial for understanding the derivation of the new algorithm for the reduction of the generalized eigenvalue problem to standard form.

1.2 Overview

As mentioned by Hendrickson, the majority of the time in an efficient eigensolver is spent in dense linear algebra operations. Because of this fact, and the existence of an efficient parallel method for solving the symmetric EVP, the majority of this thesis focuses on achieving high performance in the necessary auxiliary routines for transforming between the generalized and standard symmetric EVPs. These transformations can easily be derived because we have required that M be SPD, and therefore we can solve for its Cholesky factor, U, such that

$$M = U^T U.$$

Substituting this factorization into Eq. (1.4),

$$K\Phi = U^T U \Phi \Lambda,$$

and if we define

$$\Phi_A = U\Phi, \tag{1.5}$$

then

$$U^{-T} K U^{-1} \Phi_A = \Phi_A \Lambda. \tag{1.6}$$

Since K is symmetric, if we form

$$A = U^{-T} K U^{-1}, \tag{1.7}$$

an operation referred to as `sygst` in LAPACK[2], then we can recast the generalized EVP in symmetric standard form as

$$A\Phi_A = \Phi_A \Lambda. \tag{1.8}$$

Defining a **flop** as a floating-point operation, our parallel method for solving the generalized EVP can then be summarized as follows:

1. Form the Cholesky factor of M, $U = \Gamma(M)$ ($\frac{1}{3}n^3$ flops[12]).
2. Form $A := U^{-T}KU^{-1}$ (n^3 flops[12]).
3. Tridiagonalize A with Householder transforms, such that $T = Q^TAQ$ ($\frac{4}{3}n^3$ flops[12]).
4. Solve the tridiagonal EVP with Algorithm PMR^3 ($\mathcal{O}(n^2)$ flops[8]).
5. Back-transform the tridiagonal eigenvectors into those of the symmetric standard form using WY transforms ($2n^3$ flops[16]).
6. Back-transform the standard form eigenvectors into those of the generalized EVP using a triangular solve with multiple right-hand sides (n^3 flops[12]).

It is generally considered good practice for serial algorithms to cast as many operations as possible in terms of routines from the Level 3 Basic Linear Algebra Subprograms (BLAS)[2]. Though it is tempting to simply design parallel algorithms in terms of parallel BLAS, which ScaLAPACK calls PBLAS, it will be shown that many intermediate data distributions can be reused in order to lower the overall communication volume. However, it is still necessary to first fully develop strategies for key operations in the parallel Level 3 BLAS in order to understand how to successfully merge them for

our parallel Cholesky factorization and `sygst`. Detailed analysis of parallel algorithms for matrix-matrix multiplication and triangular solves with multiple right-hand sides are presented in Chapters Three and Four. Afterwards, these techniques are used in the development of a parallel Cholesky factorization algorithm in Chapter Five, and the most complicated parallel routine is tackled in Chapter Six, the reduction of the generalized EVP to symmetric standard form. Again, all of the mentioned operations are shown to attain significantly higher performance than ScaLAPACK.

Chapter 2

Elemental Data Distributions

The goal of this chapter is to introduce a notational scheme that allows one to incorporate matrix distributions into algebraic equations. However, there is a significant amount of machinery that must first be put in place, and so we first informally motivate the approach without the use of parallelism.

We denote a submatrix of a matrix A by appending two sets to its label, e.g., $A^{\mathcal{X},\mathcal{Y}}$, where $\mathcal{X}$ represents a subset of row indices from A and $\mathcal{Y}$ represents a subset of column indices. Note that $A^{\mathcal{X},\mathcal{Y}}$ is in general not a contiguous submatrix of A. A superscript '$\mathbb{N}$' is used to select all of the columns or rows of the matrix, and thus $A^{\mathcal{X},\mathbb{N}}$ denotes a selection of rows from a matrix A, and $A^{\mathbb{N},\mathbb{N}} = A$. Using this convention, if $C = AB$, then

$$C^{\mathcal{X},\mathcal{Y}} = [AB]^{\mathcal{X},\mathcal{Y}}.$$

Because the matrix equation $C = AB$ can be partitioned by rows as

$$\left(\begin{array}{c} \hat{c}_0^T \\ \hline \hat{c}_1^T \\ \hline \vdots \\ \hline \hat{c}_{m-1}^T \end{array}\right) = \left(\begin{array}{c} \hat{a}_0^T \\ \hline \hat{a}_1^T \\ \hline \vdots \\ \hline \hat{a}_{m-1}^T \end{array}\right) B,$$

it also holds that

$$\hat{c}_i^T = \hat{a}_i^T B, \quad i = 0, 1, ..., m-1.$$

Thus for an arbitrary selection of rows from both C and A, which we will denote by $C^{\mathcal{X},\mathbb{N}}$ and $A^{\mathcal{X},\mathbb{N}}$,

$$C^{\mathcal{X},\mathbb{N}} = A^{\mathcal{X},\mathbb{N}}B = A^{\mathcal{X},\mathbb{N}}B^{\mathbb{N},\mathbb{N}}.$$

Similarly, if we partition C and B into columns such that c_j and b_j are respectively the jth columns of C and B, then $C = AB$ implies

$$c_j = Ab_j, \quad j = 0, 1, ..., n-1.$$

Thus, for arbitrary $\mathcal{Y} \subset [0, n)$,

$$C^{\mathbb{N},\mathcal{Y}} = AB^{\mathbb{N},\mathcal{Y}} = A^{\mathbb{N},\mathbb{N}}B^{\mathbb{N},\mathcal{Y}}.$$

It follows that $C = AB$ implies

$$C^{\mathcal{X},\mathcal{Y}} = [AB]^{\mathcal{X},\mathcal{Y}} = A^{\mathcal{X},\mathbb{N}}B^{\mathbb{N},\mathcal{Y}}.$$

An obvious benefit of this notation is that when two submatrices are multiplied together, the form of their product is specified by the first superscript of the first matrix and the second superscript of the second matrix.

2.1 The Submatrix Operator

We now begin to formalize the approach so that we may expand it to a parallel setting in a well-defined manner. First, we partition an arbitrary matrix, $A \in \mathbb{R}^{m\times n}$, into two forms,

$$A \rightarrow \left(\begin{array}{c} \hat{a}_0^T \\ \hline \hat{a}_1^T \\ \hline \cdots \\ \hline \hat{a}_{m-1}^T \end{array}\right), \text{ and } A \rightarrow \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array}\right),$$

where $\hat{a}_i^T$ and a_j are, respectively, the ith row and jth column of A. Given $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{N} \equiv \mathbb{N}_0$, and using $|\cdot|$ to denote the cardinality of a set, the submatrix operator is described by

$$\left[\,\cdot\,\right]^{\mathcal{X},\mathcal{Y}} : \mathbb{R}^{m\times n} \to \mathbb{R}^{|\mathcal{X}\cap[0,m)|\,\times\,|\mathcal{Y}\cap[0,n)|}. \tag{2.1}$$

In particular, given an input matrix, A, the operator returns the submatrix of A that has had each row, $\hat{a}_i^T$, removed when $i \notin \mathcal{X}$, and each column, a_j, removed when $j \notin \mathcal{Y}$. Thus the members of each column of A are filtered by $\mathcal{X}$, and the members of each row are filtered by $\mathcal{Y}$. For this reason, $\mathcal{X}$ and $\mathcal{Y}$ will respectively be referred to as the **column filter** and **row filter**. Three trivial but extensively used identities follow:

$$\left[A^{\mathcal{X},\mathcal{Y}}\right]^T = \left[A^T\right]^{\mathcal{Y},\mathcal{X}}, \tag{2.2}$$

$$\left[A^{\mathcal{X},\mathbb{N}}\right]^{\mathbb{N},\mathcal{Y}} = A^{\mathcal{X},\mathcal{Y}}, \tag{2.3}$$

and from our previous discussion

$$A^{\mathcal{X},\mathbb{N}}B^{\mathbb{N},\mathcal{Y}} = [AB]^{\mathcal{X},\mathcal{Y}}. \tag{2.4}$$

Another useful relation stems from a partitioned matrix-matrix multiplication. If we begin with the equation

$$AB = C,$$

where $(A, B) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n}$, and partition

$$A \to \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{k-1} \end{array}\right), \; B \to \left(\begin{array}{c} \hat{b}_0^T \\ \hline \hat{b}_1^T \\ \hline \vdots \\ \hline \hat{b}_{k-1}^T \end{array}\right),$$

where a_i represents the ith column of A and $\hat{b}_i^T$ represents the ith row of B, then

$$\sum_{i=0}^{k-1} a_i \hat{b}_i^T = C.$$

Due to the associativity of addition, if we define an arbitrary partitioning of $[0,k) \subset \mathbb{N}$ into p sets, such that

$$\begin{aligned} \mathcal{Z}^q \cap \mathcal{Z}^s &= \emptyset,\ q \neq s,\ \text{and} \\ \bigcup_{q=0\ldots p-1} \mathcal{Z}^q &= [0,k)\,, \end{aligned}$$

then

$$\sum_{q=0}^{p-1} A^{\mathbb{N},\mathcal{Z}^q} B^{\mathcal{Z}^q,\mathbb{N}} = C,$$

and it follows that

$$\left[\sum_{q=0}^{p-1} A^{\mathbb{N},\mathcal{Z}^q} B^{\mathcal{Z}^q,\mathbb{N}}\right]^{\mathcal{X},\mathcal{Y}} = [C]^{\mathcal{X},\mathcal{Y}}\,,$$

and therefore

$$\sum_{q=0}^{p-1} A^{\mathcal{X},\mathcal{Z}^q} B^{\mathcal{Z}^q,\mathcal{Y}} = C^{\mathcal{X},\mathcal{Y}}. \tag{2.5}$$

Because the submatrix operator makes use of intersections of the column and row filters with the original sets of column and row numbers, the requirements for Eq. (2.5) can be loosened to

$$\begin{aligned} \mathcal{Z}^q \cap \mathcal{Z}^s &= \emptyset,\ q \neq s,\ \text{and} \\ \bigcup_{q=0\ldots p-1} \mathcal{Z}^q &= \mathcal{L} \supseteq [0,k)\,. \end{aligned}$$

Eqs. (2.2)-(2.5) are the primary tools used in later parallelizations, where we standardize on $\mathcal{L} = \mathbb{N}$.

2.2 Data Distributions

It has been theoretically shown[22] that a two-dimensional data distribution is necessary for high performance across a wide range of linear algebra operations. The key concept for 2D distributions is the **process grid**. Given a set of p processes, which are not necessarily tied to specific processors or cores, the process grid is defined as a logical $r \times c$ mesh where $(r, c) \in \mathbb{N} \times \mathbb{N}$ and $p = rc$. We choose to standardize on coordinating all equations in a column-major ordering of the process grid. For example, a 2×3 process grid is labeled

0	2	4
1	3	5

or equivalently in two-dimensional coordinates as

$(0,0)$	$(0,1)$	$(0,2)$
$(1,0)$	$(1,1)$	$(1,2)$

Three sets that repeatedly appear are the **column set**, **row set**, and **world set**,

$$\mathcal{T}_C = [0, r) \subset \mathbb{N}, \tag{2.6}$$

$$\mathcal{T}_R = [0, c) \subset \mathbb{N}, \text{ and} \tag{2.7}$$

$$\mathcal{T}_W = [0, p) \subset \mathbb{N}. \tag{2.8}$$

$\mathcal{T}_C$ provides the collection of row indices for the process grid, $\mathcal{T}_R$ similarly provides the collection of column indices, and $\mathcal{T}_W$ is simply the set of all process indices.

Perhaps the most important sets to understand are the **process rows** and **process columns**, which are respectively the rows and columns of the process grid. The motivating idea is for each process to limit its collective communications to within its own process row and process column whenever possible. Using a column-major ordering of the process grid, we define the process row and process column teams of a process whose grid coordinates are $(s,t) \in \mathcal{T}_C \times \mathcal{T}_R$ as

$$\widehat{\mathcal{T}_R^s} = \{s, s+r, ..., s+(c-1)r\} \subseteq \mathcal{T}_W, \text{ and} \tag{2.9}$$

$$\widehat{\mathcal{T}_C^t} = \{rt, rt+1, ..., rt+(r-1)\} \subseteq \mathcal{T}_W. \tag{2.10}$$

Thus $\widehat{\mathcal{T}_R^s}$ is the sth row of the process grid and $\widehat{\mathcal{T}_C^t}$ is the tth column of the process grid.

While the chosen naming conventions may seem unnatural – for instance, naming the set of row-indices the column set, $\mathcal{T}_C$ – it is done in anticipation of the algorithms in subsequent chapters. MPI collective communication routines make use of teams of processes, and thus using the chosen naming convention reinforces the connection to communicators comprised of process rows and process columns. For instance, $\mathcal{T}_C$ provides the set of local ranks in a process column communicator, and $\widehat{\mathcal{T}_C}$ provides the set of global ranks.

2.2.1 Matrix Distributions

Using slightly modified PLAPACK terminology, whenever the rows or columns of matrix data are distributed within columns of the process grid, we

say that the matrix is **column projected**. Thus, using our model 2×3 process grid, a column projection distributes either the rows or columns of a matrix between $\widehat{\mathcal{T}_R^0}$ and $\widehat{\mathcal{T}_R^1}$. In order to define this distribution for arbitrary numbers of rows and columns, we define a partitioning of $\mathbb{N}$ where each member of the partitioning is referred to as a **column projection filter**. The column projection filter for every member of process row q is given by

$$\mathcal{C}^q = \{n \in \mathbb{N} \ : \ n \bmod r = \sigma_{\mathcal{C}}^q\}, \tag{2.11}$$

where $\sigma_{\mathcal{C}}^q \in \mathcal{T}_C$ is an alignment parameter for the column projection that is particular to process row q. To ensure that the filters are a partitioning of $\mathbb{N}$ over the column set, every process row must have a unique alignment parameter in $\mathcal{T}_C$. For simplicity, we choose to order the unique parameters cyclically over $\mathcal{T}_C$, such that they satisfy the recurrence

$$\sigma_{\mathcal{C}}^{(q+1) \bmod r} = (\sigma_{\mathcal{C}}^q + 1) \bmod r. \tag{2.12}$$

Similarly, we define a **row projection** using a collection of **row projection filters**,

$$\mathcal{R}^q = \{n \in \mathbb{N} \ : \ n \bmod c = \sigma_{\mathcal{R}}^q\}, \tag{2.13}$$

where $q \in \mathcal{T}_R$ represents a particular process column number, and $\sigma_{\mathcal{R}}^q \in \mathcal{T}_R$ is an alignment parameter for the row projections specific to process column q. To again ensure that the sets constitute a partitioning of $\mathbb{N}$ over $\mathcal{T}_R$, each process column must have a unique alignment parameter in $\mathcal{T}_R$. Choosing the alignment parameters to increase cyclically within each process row, then

$$\sigma_{\mathcal{R}}^{(q+1) \bmod c} = (\sigma_{\mathcal{R}}^q + 1) \bmod c. \tag{2.14}$$

Then by construction,

$$\bigcup_{q \in \mathcal{T}_C} \mathcal{C}^q = \mathbb{N}, \text{ and} \tag{2.15}$$

$$\bigcup_{q \in \mathcal{T}_R} \mathcal{R}^q = \mathbb{N}, \tag{2.16}$$

and therefore each process can perform an Allgather within its process column to collect an entire matrix row or column that was in a column distribution, or an Allgather within its process row gathers an entire matrix row or column in a row distribution. Illustrations of the relevant collective communications are given in Fig. 2.1.

The motivation for establishing the presented notation is to *simplify* the derivation of parallel algorithms. In order to do so, we restrict ourselves to synchronous parallel algorithms so that the actions of each process can be described equivalently up to translations of alignment parameters for the distribution superscripts. Our restriction to synchronous algorithms is justified by our use of elemental distributions, as they result in very evenly distributed matrices, which means that load balance is typically not an issue. **From now on, when any filter appears in an equation without the superscript that specifies which process or processes it applies to, the equation applies to every permissible choice of the superscript for the entire equation.** For instance, stating

$$[AB]^{\mathcal{C},\mathcal{R}} = C^{\mathcal{C},\mathcal{R}}$$

Operation	Before				After			
Send-recv ring	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x_0					x_0		
		x_1					x_1	
			x_2					x_2
				x_3	x_3			
Allgather	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	x_0				x_0	x_0	x_0	x_0
		x_1			x_1	x_1	x_1	x_1
			x_2		x_2	x_2	x_2	x_2
				x_3	x_3	x_3	x_3	x_3
Reduce-scatter	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	$x_0^{(0)}$	$x_0^{(1)}$	$x_0^{(2)}$	$x_0^{(3)}$	$\sum_j x_0^{(j)}$			
	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$		$\sum_j x_1^{(j)}$		
	$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$			$\sum_j x_2^{(j)}$	
	$x_3^{(0)}$	$x_3^{(1)}$	$x_3^{(2)}$	$x_3^{(3)}$				$\sum_j x_3^{(j)}$
All-to-all	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
	$x_0^{(0)}$	$x_0^{(1)}$	$x_0^{(2)}$	$x_0^{(3)}$	$x_0^{(0)}$	$x_1^{(0)}$	$x_2^{(0)}$	$x_3^{(0)}$
	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$
	$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$
	$x_3^{(0)}$	$x_3^{(1)}$	$x_3^{(2)}$	$x_3^{(3)}$	$x_0^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$

Figure 2.1: Overview of relevant collective communication patterns. For the demonstration of a Send-recv ring, each node is sending to the node to its right and receiving from the node to its left. This technique is used to transition between multivector distributions and is discussed at the end of the chapter.

becomes equivalent to writing

$$[AB]^{\mathcal{C}^s,\mathcal{R}^t} = C^{\mathcal{C}^s,\mathcal{R}^t}, \quad \forall\,(s,t) \in \mathcal{T}_C \times \mathcal{T}_R.$$

Using this abstraction, we have effectively homogenized the description of each process's work and can concisely describe the work of the entire set of processes using a single equation.

We now define the **matrix distribution operator**,

$$[\,\cdot\,]^{\mathcal{C},\mathcal{R}} : \mathbb{R}^{m\times n} \to \mathbb{R}^{|\mathcal{C}\cap[0,m)| \,\times\, |\mathcal{R}\cap[0,n)|}, \tag{2.17}$$

as the specific case of the submatrix operator where the column and row filters are chosen to be the column and row projection filters. Similarly, we define the **transposed matrix distribution operator**,

$$[\,\cdot\,]^{\mathcal{R},\mathcal{C}} : \mathbb{R}^{m\times n} \to \mathbb{R}^{|\mathcal{R}\cap[0,m)| \,\times\, |\mathcal{C}\cap[0,n)|}, \tag{2.18}$$

as the case where the column and row filters are respectively row and column projection filters. In other words, columns of the matrix are distributed over rows of the process grid, and vice versa. Using the 2×3 process grid, the resulting matrix and transposed matrix distributions of a 7×7 matrix for two different choices of alignment parameters are shown in Table 2.1. In a matrix distribution, $(\sigma^s_{\mathcal{C}}, \sigma^t_{\mathcal{R}})$ provides the global indices of the upper left element of the local matrix on process (s,t). In a transposed matrix distribution, $(\sigma^t_{\mathcal{R}}, \sigma^s_{\mathcal{C}})$ serves the same purpose. These distributions are respectively described by tiling the process grid and transposed process grid over the matrix. Given $(\sigma^s_{\mathcal{C}}, \sigma^t_{\mathcal{R}})$ for any (s,t), the tiling is fully specified.

	matrix dist.	transposed matrix dist.
$(\sigma^0_{\mathcal{C}}, \sigma^0_{\mathcal{R}}) = (0,0)$	$\begin{pmatrix} 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 \\ 4 & 5 & 4 & 5 & 4 & 5 & 4 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 \\ 4 & 5 & 4 & 5 & 4 & 5 & 4 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$
$(\sigma^0_{\mathcal{C}}, \sigma^0_{\mathcal{R}}) = (1,1)$	$\begin{pmatrix} 5 & 1 & 3 & 5 & 1 & 3 & 5 \\ 4 & 0 & 2 & 4 & 0 & 2 & 4 \\ 5 & 1 & 3 & 5 & 1 & 3 & 5 \\ 4 & 0 & 2 & 4 & 0 & 2 & 4 \\ 5 & 1 & 3 & 5 & 1 & 3 & 5 \\ 4 & 0 & 2 & 4 & 0 & 2 & 4 \\ 5 & 1 & 3 & 5 & 1 & 3 & 5 \end{pmatrix}$	$\begin{pmatrix} 5 & 4 & 5 & 4 & 5 & 4 & 5 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 3 & 2 & 3 & 2 & 3 & 2 & 3 \\ 5 & 4 & 5 & 4 & 5 & 4 & 5 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 3 & 2 & 3 & 2 & 3 & 2 & 3 \\ 5 & 4 & 5 & 4 & 5 & 4 & 5 \end{pmatrix}$

Table 2.1: Overlays of the owning process of each element of a seven-by-seven matrix in four different matrix distributions.

We also define **partial matrix distributions** (PMDs) to be cases in which only one of the filters of the submatrix operator is chosen from $\{\mathcal{C}, \mathcal{R}\}$, and the other is $\mathbb{N}$. The name *partial* matrix distribution was chosen because any matrix distribution can be described using a combination of two partial matrix distribution operators. For instance, we can express a standard matrix distribution operator as a combination of a column-projected column-wise PMD and a row-projected row-wise PMD:

$$[\,\cdot\,]^{\mathcal{C},\mathcal{R}} = \Big[[\,\cdot\,]^{\mathcal{C},\mathbb{N}}\Big]^{\mathbb{N},\mathcal{R}},$$

where the operator $[\,\cdot\,]^{\mathcal{C},\mathbb{N}}$ is described as column-projected because the set $\mathcal{C}$ represents a *distribution among process columns*. It is called column-wise

$[\,\cdot\,]^{\mathcal{C},\mathbb{N}}$	Column-projected column-wise
$[\,\cdot\,]^{\mathbb{N},\mathcal{C}}$	Column-projected row-wise
$[\,\cdot\,]^{\mathcal{R},\mathbb{N}}$	Row-projected column-wise
$[\,\cdot\,]^{\mathbb{N},\mathcal{R}}$	Row-projected row-wise

Table 2.2: The four partial matrix distribution (PMD) operator categories.

because it applies the column filter to each *column* of the matrix. Similarly, $[\,\cdot\,]^{\mathbb{N},\mathcal{R}}$ is a row-projected row-wise PMD operator. It is row-projected because $\mathcal{R}$ represents a *distribution among process rows*, and it is row-wise because $\mathcal{R}$ is applied to the *rows* of the matrix.

PMDs are used heavily in the later algorithms for parallel matrix-matrix multiplication, and the four categories are listed in Table 2.2. Though PLAPACK refers to partial matrix distributions as **duplicated projected multivectors**, their use in algorithms is best described by their connection to matrix distributions. Several examples of partial matrix distributions are shown in Table 2.3.

2.2.2 Multiscalars and Multivectors

We refer to a one-dimensional distribution among all processes as a **multivector distribution** (MVD), which corresponds to only distributing either the columns or rows of a matrix, but doing so over the entire process grid. A PMD also distributes either columns or rows of a matrix, but only over the process rows or process columns instead of the global set of processes. The

	column-wise	row-wise
column-projected	$\begin{pmatrix} \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} \\ \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} \\ \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} \\ \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^1} \\ \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^0} \end{pmatrix}$	$\begin{pmatrix} \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} \\ \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} \\ \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} \\ \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} \\ \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} & \widehat{\mathcal{T}_R^1} & \widehat{\mathcal{T}_R^0} \end{pmatrix}$
row-projected	$\begin{pmatrix} \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} \\ \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} \\ \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^2} \\ \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^0} \\ \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^1} \end{pmatrix}$	$\begin{pmatrix} \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} \\ \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} \\ \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} \\ \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} \\ \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} & \widehat{\mathcal{T}_C^2} & \widehat{\mathcal{T}_C^0} & \widehat{\mathcal{T}_C^1} \end{pmatrix}$

Table 2.3: Overlays of the process rows or columns associated with individual elements of a five-by-five matrix in four different partial matrix distributions, with $\sigma_{\mathcal{C}}^0 = \sigma_{\mathcal{R}}^0 = 0$.

concept of multivectors originates from Physically Based Matrix Distributions, and we motivate their use through the system of equations

$$UX = B = \left(\; b_0 \; \middle| \; b_1 \; \middle| \; \cdots \; \middle| \; b_{n-1} \; \right),$$

where $U \in \mathbb{R}^{m \times m}$ is upper-triangular, $X \in \mathbb{R}^{m \times n}$ is the desired solution, and b_i is the ith column of the right-hand side. We define a **multiscalar** to be a matrix that is fully owned by every process, which corresponds to the case where the submatrix filters are both chosen to be $\mathbb{N}$. If U is chosen to be a multiscalar, and B is stored in a row-wise multivector, meaning that each column of B is stored on a single process, then trivial parallelism can be achieved in solving for X. Each process can simply perform a local **tr**iangle **s**olve with **m**ultiple right-hand sides (**trsm**) using U and the local set of columns of B, and high performance can be expected as long as $n \gg p$.

Because the cost of moving between multivector and matrix distributions cannot simply be ignored, we define two special cases of multivectors with the goal of reducing the cost of transitions. The first is a **column-major multivector**, whose distribution is specified by applying to each process q, a set

$$\mathcal{M}_C^q = \left\{ m \in \mathbb{N} \; : \; m \bmod p = \sigma^q_{\mathcal{M}_C} \right\} \tag{2.19}$$

as one of the filters in the submatrix operator, and setting the other filter as $\mathbb{N}$. The distinguishing characteristic for each multivector is the recurrence used to constrain the alignment parameters. For column-major multivectors,

we require that

$$\sigma_{\mathcal{M}_C}^{(q+1) \bmod p} = \left(\sigma_{\mathcal{M}_C}^{q} + 1\right) \bmod p. \tag{2.20}$$

The purpose of this constraint is to require that the alignment parameters increase (mod p) as one wraps around the process grid in a column-major fashion. Because we have used a column-major ordering to label the process grid, the above constraint simply requires that the alignment parameters cyclically increase as the process indices increase. If we had chosen to standardize on a row-major labeling of the process grid, the expression for the constraint would become more complicated.

A **row-major multivector** can be thought of as the dual to the column-major multivector, as it has each process, $q \in \mathcal{T}_W$, use the set

$$\mathcal{M}_R^q = \left\{ m \in \mathbb{N} \ : \ m \bmod p = \sigma_{\mathcal{M}_R}^q \right\}, \tag{2.21}$$

where $\sigma_{\mathcal{M}_R}^q \in \mathcal{T}_W$ is required to increase (mod p) as one wraps around the process grid in a *row-major* fashion. If we were to use a row-major labeling for our process grid, the constraints on $\left\{\sigma_{\mathcal{M}_R}^q\right\}$ would become identical to those on $\left\{\sigma_{\mathcal{M}_C}^q\right\}$ expressed in a column-major labeling. In order to be able to standardize on a single labeling of the process grid, it is helpful to define mappings that will transform a process's column-major label into its row-major label.

Given a process whose coordinates are (s, t), its column-major rank, q, is

$$q = s + rt,$$

and its row-major rank, $\hat{q}$, is

$$\hat{q} = t + cs.$$

Then given $q \in \mathcal{T}_W$ in column-major ordering we have

$$\begin{aligned} s &= q \bmod r, \text{ and} \\ t &= \left\lfloor \frac{q}{r} \right\rfloor, \end{aligned}$$

and given $\hat{q} \in \mathcal{T}_W$ in row-major ordering, its process row and column indices are respectively

$$\begin{aligned} s &= \left\lfloor \frac{\widehat{q}}{c} \right\rfloor, \text{ and} \\ t &= \widehat{q} \bmod c. \end{aligned}$$

We can then construct our mapping from a column-major label to row-major label, f, and its inverse, f^{-1}, as

$$f(q) = \left\lfloor \frac{q}{r} \right\rfloor + c\,(q \bmod r)\,, \text{ and} \tag{2.22}$$

$$f^{-1}(\widehat{q}) = \left\lfloor \frac{\widehat{q}}{c} \right\rfloor + r\,(\widehat{q} \bmod c)\,. \tag{2.23}$$

Given $q \in \mathcal{T}_W$, where q is a column-major label, one can map it to row-major ordering, cyclically increase it, then transform back to column-major ordering by the operation $f^{-1}\left((f(q)+1) \bmod p\right)$. Thus the row-major constraint in column-major labeling can be expressed as

$$\sigma_{\mathcal{M}_R}^{f^{-1}((f(q)+1) \bmod p)} = \left(\sigma_{\mathcal{M}_R}^{q} + 1\right) \bmod p.$$

Alternatively, from any diagram of a process grid we can recognize that incrementing a process's column-major label by one is equivalent to increasing its row-major label by $c \pmod p$. Thus we may alter Eq. (2.20) such that

$$\sigma_{\mathcal{M}_R}^{(q+1) \bmod p} = \left(\sigma_{\mathcal{M}_R}^{q} + c\right) \bmod p. \tag{2.24}$$

Our task is now to define a relation between the defined multivector distributions and the row and column projections. We start by looking at the connection between column-major multivectors and column projections. For each $s \in \mathcal{T}_C$, there exists a unique $x \in \mathcal{T}_C$ such that

$$\sigma_{\mathcal{C}}^{x} = \sigma_{\mathcal{M}_C}^{q} \bmod r, \quad \forall q \in \widehat{\mathcal{T}_R^s}, \tag{2.25}$$

because the alignment parameters for column-major multivectors increase cyclically, mod p, down the process columns of length r. Thus $\sigma_{\mathcal{M}_C}^{q} \bmod r$ does not change within process rows. Keeping the same pair (s, x), it follows that

$$\bigcup_{q \in \widehat{\mathcal{T}_R^s}} \mathcal{M}_C^q = \mathcal{C}^x, \tag{2.26}$$

which shows that an Allgather within each process row can redistribute a column-major multivector distribution, $[\,\cdot\,]^{\mathcal{M}_C^q,\mathbb{N}}$ or $[\,\cdot\,]^{\mathbb{N},\mathcal{M}_C^q}$, into a column-projected PMD, respectively $[\,\cdot\,]^{\mathcal{C}^x,\mathbb{N}}$ or $[\,\cdot\,]^{\mathbb{N},\mathcal{C}^x}$.

Similarly, for each $t \in \mathcal{T}_R$, there exists a unique $y \in \mathcal{T}_R$ such that

$$\sigma_{\mathcal{R}}^{y} = f\left(\sigma_{\mathcal{M}_R}^{q}\right) \bmod c, \quad \forall q \in \widehat{\mathcal{T}_C^t}, \tag{2.27}$$

because the alignment parameters for row-major multivectors increase cyclically, mod p, within the process rows of length c. Therefore $f\left(\sigma_{\mathcal{M}_R}^{q}\right) \bmod c$

is invariant within process columns. Using the same pair (t, y), it follows that

$$\bigcup_{q \in \widehat{\mathcal{T}_C^t}} \mathcal{M}_R^q = \mathcal{R}^y, \tag{2.28}$$

which indicates that an Allgather within each process column can transform a row-major multivector distribution, $[\,\cdot\,]^{\mathcal{M}_R^q,\mathbb{N}}$ or $[\,\cdot\,]^{\mathbb{N},\mathcal{M}_R^q}$ into a row-projected PMD, respectively $[\,\cdot\,]^{\mathcal{R}^y,\mathbb{N}}$ or $[\,\cdot\,]^{\mathbb{N},\mathcal{R}^y}$.

Wrapping around a diagonal of a tiling of the process grid can be thought of as simultaneously wrapping around the rows and columns of the process grid, and ideally we could define a single multivector that could replace both column-major and row-major multivectors. Because a multivector is required to span the entire set of processes, a simultaneous wrapping can only be achieved when the least common multiple (**lcm**) of (r, c) is $p = rc$, since the period of a diagonal wrapping will be the **lcm** of r and c. It is well known that

$$\operatorname{lcm}(a, b) = \frac{ab}{\gcd(a, b)},$$

where we are also using the greatest common divisor (**gcd**) of (a, b). Two integers are called **coprime** if their **gcd** is 1. Thus $\operatorname{lcm}(r, c) = rc = p$ if and only if r and c are coprime, and we can only define a diagonal wrapping when $\gcd(r, c) = 1$. We refer to the resulting distribution as a **diagonal multivector**, which is partly described by each process q using the filter

$$\mathcal{M}_D^q = \left\{ m \in \mathbb{N} \;:\; m \bmod p = \sigma_{\mathcal{M}_D}^q \right\}. \tag{2.29}$$

In order to describe the constraints on $\left\{\sigma_{\mathcal{M}_D}^q\right\}$, we will introduce a slight modification to the current notation. Since a process's label may be specified

by its process grid coordinates, $(s,t) \in \mathcal{T}_C \times \mathcal{T}_R$, we may write a column-major label q as

$$q = q(s,t) = s + tr.$$

Then

$$\begin{aligned} \mathcal{M}_D^{q(s,t)} &= \mathcal{M}_D^{s+tr}, \text{ and} \\ \sigma_{\mathcal{M}_D}^{q(s,t)} &= \sigma_{\mathcal{M}_D}^{s+tr} \end{aligned}$$

give a potentially more expressive means of describing the relation between alignment parameters in a diagonal multivector distribution. The requirement that $\sigma_{\mathcal{M}_D}^q$ increases cyclically down a diagonal of the process grid tiling can be expressed as

$$\sigma_{\mathcal{M}_D}^{q((s+1) \bmod r,\,(t+1) \bmod c)} = \left(\sigma_{\mathcal{M}_D}^{q(s,t)} + 1\right) \bmod p. \tag{2.30}$$

Analogously to Eqs. (2.25) and (2.27), when r and c are coprime, for each $s \in \mathcal{T}_C$, there exists a unique $x \in \mathcal{T}_C$ such that

$$\sigma_{\mathcal{C}}^x = \sigma_{\mathcal{M}_D}^q \bmod r, \quad \forall q \in \widehat{\mathcal{T}_R^s}, \tag{2.31}$$

and for each $t \in \mathcal{T}_R$, there exists a unique $y \in \mathcal{T}_R$ such that

$$\sigma_{\mathcal{R}}^y = f\left(\sigma_{\mathcal{M}_D}^q\right) \bmod c, \quad \forall q \in \widehat{\mathcal{T}_C^t}. \tag{2.32}$$

Then, for each $q(s,t) \in \mathcal{T}_W$ there exists a unique $(x,y) \in \mathcal{T}_C \times \mathcal{T}_R$ such that

$$\bigcup_{q \in \widehat{\mathcal{T}_R^s}} \mathcal{M}_D^q = \mathcal{C}^x, \tag{2.33}$$

$$\bigcup_{q \in \widehat{\mathcal{T}_C^t}} \mathcal{M}_D^q = \mathcal{R}^y, \text{ and} \tag{2.34}$$

$$\mathcal{M}_D^q = \mathcal{C}^x \cap \mathcal{R}^y. \tag{2.35}$$

$[\,\cdot\,]^{\mathcal{M}_C,\mathbb{N}}$	Column-major column-wise
$[\,\cdot\,]^{\mathbb{N},\mathcal{M}_C}$	Column-major row-wise
$[\,\cdot\,]^{\mathcal{M}_R,\mathbb{N}}$	Row-major column-wise
$[\,\cdot\,]^{\mathbb{N},\mathcal{M}_R}$	Row-major row-wise
$[\,\cdot\,]^{\mathcal{M}_D,\mathbb{N}}$	Diagonal column-wise
$[\,\cdot\,]^{\mathbb{N},\mathcal{M}_D}$	Diagonal row-wise

Table 2.4: The six multivector distribution (MVD) categories.

Eqs. (2.33) and (2.34) demonstrate that a diagonal multivector can be redistributed as a column-projected PMD using an Allgather within process rows, and redistributed as a row-projected PMD using an Allgather within process columns. The six categories of multivectors that arise from column-major, row-major, and diagonal multivector sets are shown in Table 2.4, and overlays of their distributions are shown in Table 2.5.

2.3 Transposing Distributions

In the previous section, it was shown how to perform several straightforward redistributions that require only a call to the MPI collective Allgather. However, it is often the case that an algorithm requires either a complete transposition of a distribution or something similar, such as redistributing a matrix A from the form $A^{\mathcal{C},\mathcal{R}}$ to $A^{\mathcal{R},\mathcal{C}}$ or $A^{\mathcal{R},\mathbb{N}}$. In general, these redistributions cannot be efficiently performed using a single MPI routine. We provide a brief explanation of the modeled costs of several MPI routines and then give an overview

	column-wise	row-wise
column-major	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}$
row-major	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 4 & 1 & 3 & 5 \\ 0 & 2 & 4 & 1 & 3 & 5 \\ 0 & 2 & 4 & 1 & 3 & 5 \\ 0 & 2 & 4 & 1 & 3 & 5 \\ 0 & 2 & 4 & 1 & 3 & 5 \\ 0 & 2 & 4 & 1 & 3 & 5 \end{pmatrix}$
diagonal	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$	$\begin{pmatrix} 0 & 3 & 4 & 1 & 2 & 5 \\ 0 & 3 & 4 & 1 & 2 & 5 \\ 0 & 3 & 4 & 1 & 2 & 5 \\ 0 & 3 & 4 & 1 & 2 & 5 \\ 0 & 3 & 4 & 1 & 2 & 5 \\ 0 & 3 & 4 & 1 & 2 & 5 \end{pmatrix}$

Table 2.5: Overlays of the owning process of each element of a six-by-six matrix in six different multivector distributions, where $\sigma^0_{\mathcal{M}_C} = \sigma^0_{\mathcal{M}_R} = \sigma^0_{\mathcal{M}_D} = 0$.

of the steps involved in distribution transposition. Understanding the process is crucial as it highlights many of the redistribution issues that arise in the algorithms in later chapters.

2.3.1 Communication Costs

There are several commonly used models for collective communication costs, such as LogP/LogGP and PLogP, but we choose to focus on a modified version of the Hockney model[21]. It assumes that the time to send a single message is $t_m = \alpha + \beta n$, where α is the latency of sending a message, β is the inverse of the network bandwidth, and n is the message size. Computational cost is separately considered using $t_c = \gamma d$, where γ represents the cost in time of performing a single operation, typically addition, and d is the number of operations performed.

MPI implementations typically minimize latency for "small" messages and minimize communication volume for "large" messages[23], so it is reasonable to assume that a good algorithm will have a cost similar to the latency lower bound for small messages, and similar to the bandwidth lower bound for large messages. The line between "small" and "large" messages is typically a function of the number of MPI processes and is decided by the implementation. A list of lower bounds, from Chan et al.[6], of several common MPI operations is given in Table 2.6. Each lower bound was computed using the Hockney model under the assumption that each process can simultaneously send a message to a process and receive a message from a (possibly different)

Communication	Latency	Bandwidth	Computation
Send-recv	α	$n\beta$	–
Allgather	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–
Reduce-scatter	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$
All-to-all	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–

Table 2.6: Lower bounds for costs of select MPI routines. n is the characteristic message size for each routine.

process. It is also assumed that there is no message interference or network congestion. All-to-all lower bounds were not given by Chan et al. and were analyzed by the author using the same communication model. The lower bound on the All-to-all latency should be paid particular attention, as it requires a large increase in communication volume. A typical All-to-all algorithm involves all processes directly communicating with all other processes[23] and therefore has a latency component of $(p-1)\alpha$.

2.3.2 The General Approach

The PLAPACK strategy of transposing a distribution is to use a multivector distribution as an intermediate step. Our approach is analogous, though we are in general required to use two different multivector distributions in the process rather than just one. The concept of **distribution blocks** is helpful for understanding why. If one is to determine a matrix distribution through a tiling of the process grid over a matrix, in general each member of the process grid could represent an $m_b \times n_b$ block of matrix elements called a distribution block. The elemental approach to matrix distributions is defined by the

choice of 1×1 distribution blocks. PLAPACK requires that $n_b = rm_b$, where r is again the number of process rows in the logical process grid. By skewing the process grid tiling with distribution blocks that are r times wider than their height, the diagonal of the process grid tiling will always span the entire set of processes. For this reason, PLAPACK can use a multivector distribution analogous to our diagonal multivector. However, skewing the distribution blocks to have an aspect ratio equal to the number of process rows is inherently unscalable.

We begin by describing the approach to transposing a distribution when the diagonal multivector distribution exists, and we use '$\Rightarrow$' to denote the act of redistributing, which consists of packing, communicating, and unpacking data. The transposition sequence, shown at the top of Fig. 2.2, can be summarized as

$$A^{\mathcal{C},\mathcal{R}} \Rightarrow A^{\mathcal{M}_D,\mathbb{N}} \Rightarrow A^{\mathcal{R},\mathcal{C}},$$

where, for each process $q(s,t)$, $\sigma^q_{\mathcal{M}_D}$ is chosen so that

$$\mathcal{M}^q_D \equiv \mathcal{C}^s \cap \mathcal{R}^t.$$

The first step,

$$A^{\mathcal{C}^s,\mathcal{R}^t} \Rightarrow A^{\mathcal{M}^q_D,\mathbb{N}}, \quad \forall\ (s,t) \in \mathcal{T}_C \times \mathcal{T}_R,$$

requires communication that involves collecting columns, indicated by the transition of the second superscript from $\mathcal{R}^t$ to $\mathbb{N} \supset \mathcal{R}^t$, but only needing a subset of the originally owned rows, indicated by the change from $\mathcal{C}^s$ to $\mathcal{M}^q_D \subset \mathcal{C}^s$. This selective collection of subsets of columns can be accomplished

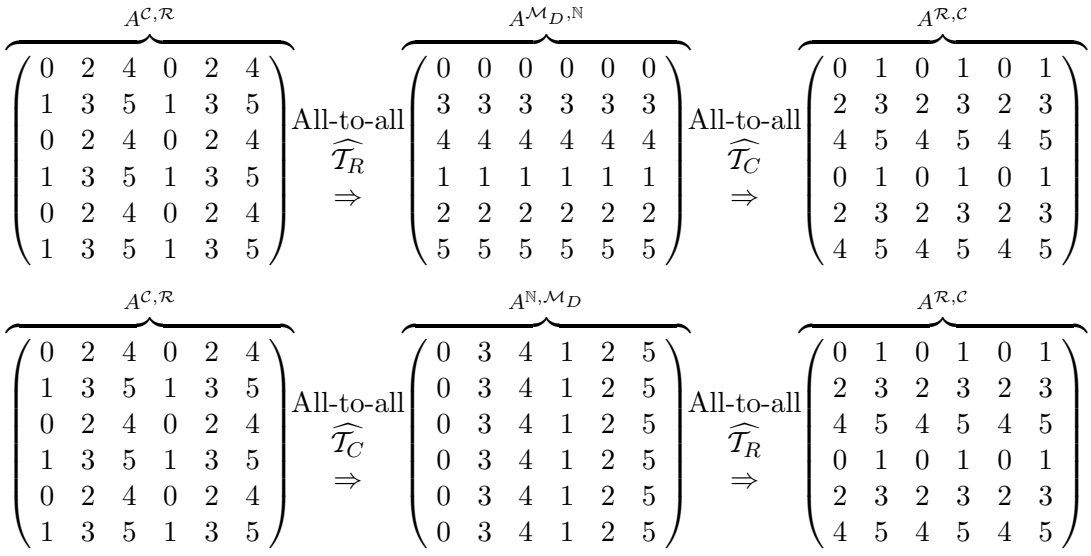

Figure 2.2: Overlays of the owning process of each element of a 6×6 matrix during two different distribution transposition methods.

by an All-to-all within process rows. A simpler, but much less efficient, alternative would be to perform an Allgather within process rows and discard the unneeded rows of the gathered data. The next step,

$$A^{\mathcal{M}_D^q,\mathbb{N}} \Rightarrow A^{\mathcal{R}^t,\mathcal{C}^s}, \quad \forall\ (s,t) \in \mathcal{T}_C \times \mathcal{T}_R,$$

is analogous to the first step, and can be accomplished via an All-to-all within process columns.

It is important to realize that the distribution transposition could also be achieved with the sequence

$$A^{\mathcal{C},\mathcal{R}} \Rightarrow A^{\mathbb{N},\mathcal{M}_D} \Rightarrow A^{\mathcal{R},\mathcal{C}},$$

which is shown at the bottom of Fig. 2.2. While the choice is arbitrary for simply transposing the distribution, if we instead wanted to redistribute $A^{\mathcal{C},\mathcal{R}}$

to $A^{\mathbb{N},\mathcal{C}}$, the second approach is advantageous to the first. The naïve extension would be the sequence

$$A^{\mathcal{C},\mathcal{R}} \Rightarrow A^{\mathbb{N},\mathcal{M}_D} \Rightarrow A^{\mathcal{R},\mathcal{C}} \Rightarrow A^{\mathbb{N},\mathcal{C}},$$

where the last communication is performed using an Allgather within process rows. Because the last two communication steps are within process rows, we can simply combine them into a single communication and perform the sequence

$$A^{\mathcal{C},\mathcal{R}} \Rightarrow A^{\mathbb{N},\mathcal{M}_D} \Rightarrow A^{\mathbb{N},\mathcal{C}}.$$

From Eq. (2.33) we know that the last communication can be performed with an Allgather within each process row. The process is demonstrated in Fig. 2.3.

For general process grids, we cannot assume that r and c are coprime, so we must be able to handle distribution transposition without making use of the diagonal multivector distribution. We are well equipped because of our definitions of column-major and row-major multivectors. In fact, the only difference from the approach with diagonal multivectors is that there is an additional redistribution between column-major and row-major multivectors. For instance, if our process grid is

0	2	4	6
1	3	5	7

the sequence

$$A^{\mathcal{C},\mathcal{R}} \Rightarrow A^{\mathcal{M}_D,\mathbb{N}} \Rightarrow A^{\mathcal{R},\mathcal{C}},$$

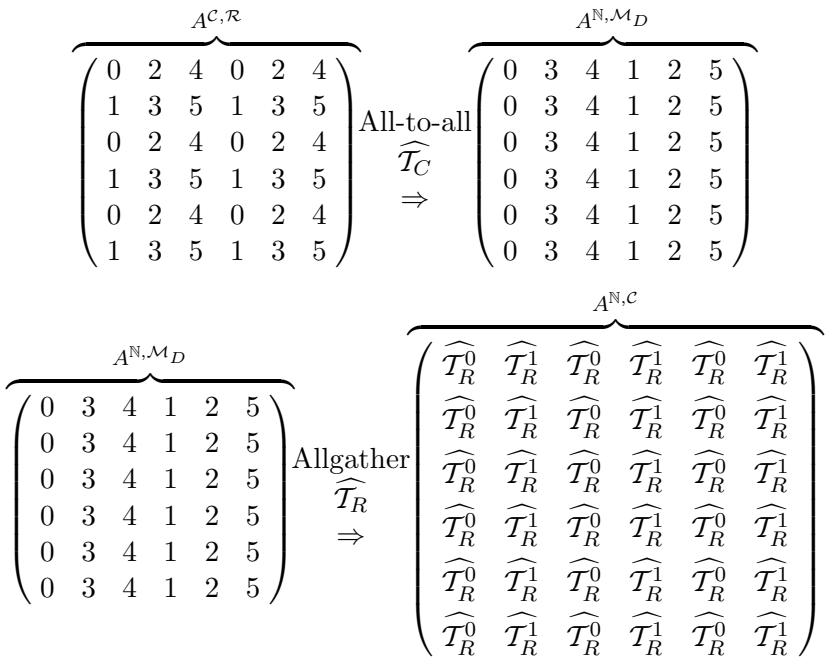

Figure 2.3: Overlays of the owning processes of each element of a 6×6 matrix during the redistribution from a matrix distribution to a column-projected row-wise PMD. $\widehat{\mathcal{T}_R^0} = \{0, 2, 4\}$, and $\widehat{\mathcal{T}_R^1} = \{1, 3, 5\}$.

is replaced with

$$A^{\mathcal{C},\mathcal{R}} \Rightarrow A^{\mathcal{M}_C,\mathbb{N}} \Rightarrow A^{\mathcal{M}_R,\mathbb{N}} \Rightarrow A^{\mathcal{R},\mathcal{C}},$$

shown in Fig. 2.4, where $\sigma_{\mathcal{M}_C}^{q(s,t)}$ is chosen so that $\sigma_{\mathcal{M}_C}^{q(s,t)} \bmod r = \sigma_{\mathcal{C}}^{s}$ for all $t \in \mathcal{T}_R$, and $\sigma_{\mathcal{M}_R}^{q(s,t)}$ is chosen so that $f\left(\sigma_{\mathcal{M}_R}^{q(s,t)}\right) \bmod c = \sigma_{\mathcal{R}}^{t}$ for all $s \in \mathcal{T}_C$. The first and last communications are nearly unchanged, but we have the added complication of the step

$$A^{\mathcal{M}_C^q,\mathbb{N}} \Rightarrow A^{\mathcal{M}_R^q,\mathbb{N}}, \quad \forall\, (s,t) \in \mathcal{T}_C \times \mathcal{T}_R,$$

which can be accomplished with a single call to Send-recv on each process. The redistribution is trivial because the only differences between column-major and row-major multivectors are the alignment parameters. Additionally, Table 2.6 indicates that a Send-recv has a very low cost. Redistributing between multivectors is a matter of each process receiving from the process that owns in the current distribution what it would own in the new distribution, and simultaneously sending to the process that will own in the new distribution what it currently owns. More precisely, given a process $q(s,t)$, where

$$\Delta\sigma^q = \sigma_{\mathcal{M}_C}^q - \sigma_{\mathcal{M}_R}^q,$$

process q sends its data to process q_{send}, where

$$q_{\text{send}} = f\left((q + \Delta\sigma^q) \bmod p\right),$$

and receives its data from process q_{recv}, where

$$q_{\text{recv}} = f^{-1}\left((q - \Delta\sigma^q) \bmod p\right).$$

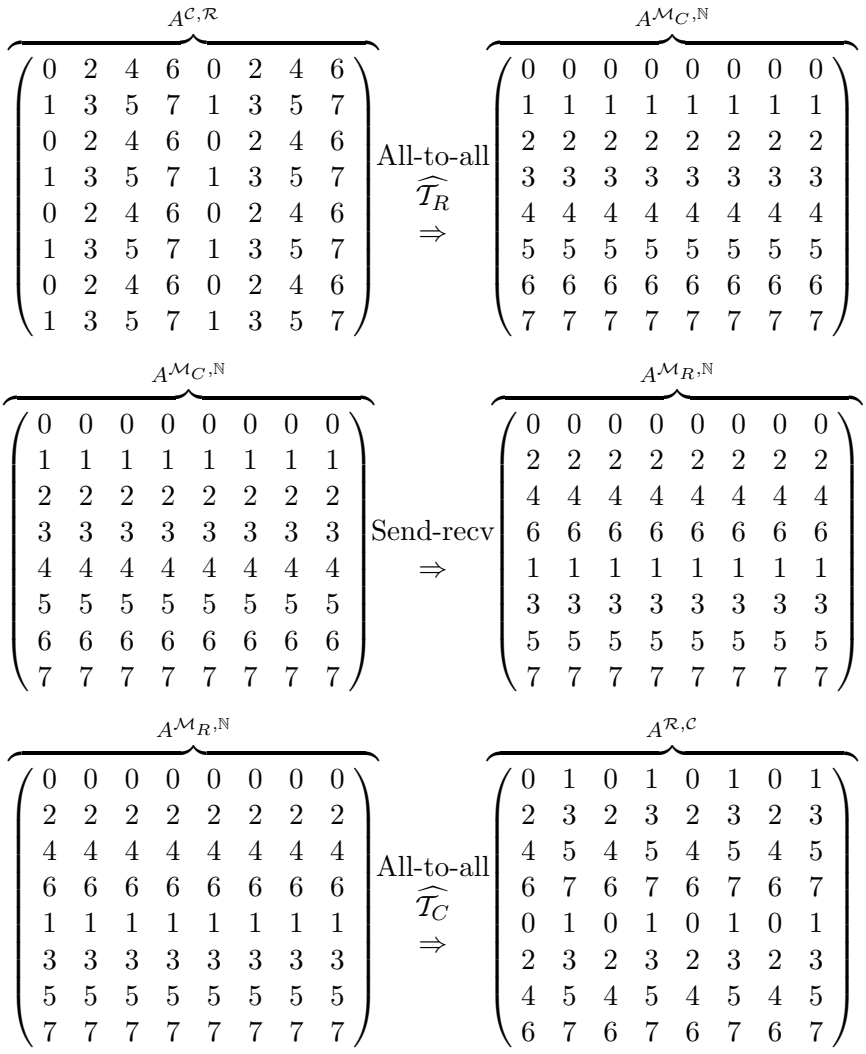

Figure 2.4: Overlays of the owning process of each element of an 8×8 matrix during a distribution transposition with a 2×4 process grid.

For convenience, from now on we will use

$$[\,\cdot\,]^{\leftrightarrow} : A^{\mathcal{X},\mathcal{Y}} \to A^{\mathcal{Y},\mathcal{X}}, \tag{2.36}$$

to represent the result of a general distribution transposition.

It should also be noted that, in the case of a square process grid and matrix, a shortcut exists for transposing the matrix distribution when $\sigma^0_{\mathcal{C}} = \sigma^0_{\mathcal{R}}$. Taking advantage of symmetry, each process, with coordinates (s,t), need only exchange local data with process (t,s) in order to complete the distribution transposition. Finally, calls to Send-recv are extremely cheap compared to the use of MPI collectives and can be used for shifting alignment parameters. In later chapters, all analysis assumes that alignment parameters of input matrices are compatible. In the general case this is not true, but the performance impact of calls to Send-recv to adjust alignment parameters is negligible and the handling of this technicality is left to implementations.

Chapter 3

Matrix-Matrix Multiplication

Matrix-matrix multiplication, the composition of linear transformations, is at the core of dense linear algebra, so understanding how to optimize its parallel performance provides a strong foundation for developing more complicated operations. Using terminology from BLAS, **ge**neral **m**atrix-**m**atrix multiplication (**gemm**) is the class of operations

$$C := \alpha \, \mathrm{op}(A) \, \mathrm{op}(B) + \beta C, \tag{3.1}$$

where $\mathrm{op}(\cdot)$ can either return the input matrix or its transpose, α and β are arbitrary scalars, $\mathrm{op}(A) \in \mathbb{R}^{m \times k}$, $\mathrm{op}(B) \in \mathbb{R}^{k \times n}$, and $C \in \mathbb{R}^{m \times n}$. For the sake of simplicity, we will assume $\alpha = 1$ and $\beta = 0$ for our analysis. The four basic categories are then

- $C := AB$ (Normal-Normal)
- $C := AB^T$ (Normal-Transposed)
- $C := A^T B$ (Transposed-Normal)
- $C := A^T B^T$ (Transposed-Transposed)

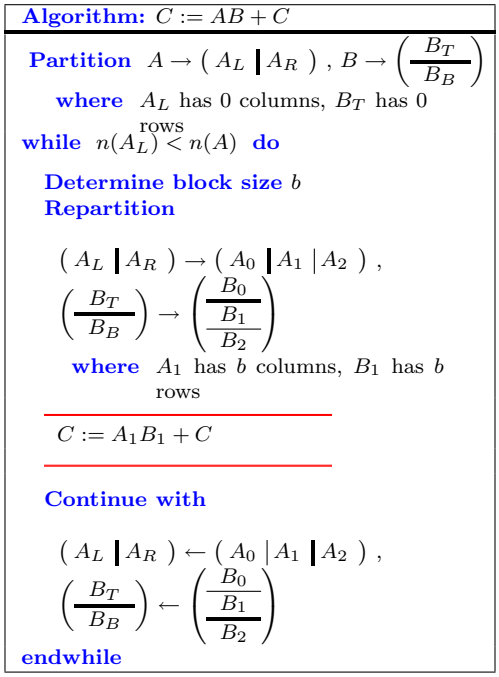

Figure 3.1: Blocked panel-panel algorithm for normal-normal **gemm**.

For distributed-memory implementations, it is common[5, 24] to also divide algorithms into the three cases

- Stationary A
- Stationary B
- Stationary C,

where the **stationary** matrix is not communicated during the algorithm. In the extreme case where $n \ll (m, k)$, which can be described as a matrix-panel multiply[13], B and C are significantly smaller than A and it is often beneficial to avoid communicating A. Similarly, when $m \ll (n, k)$, one should avoid

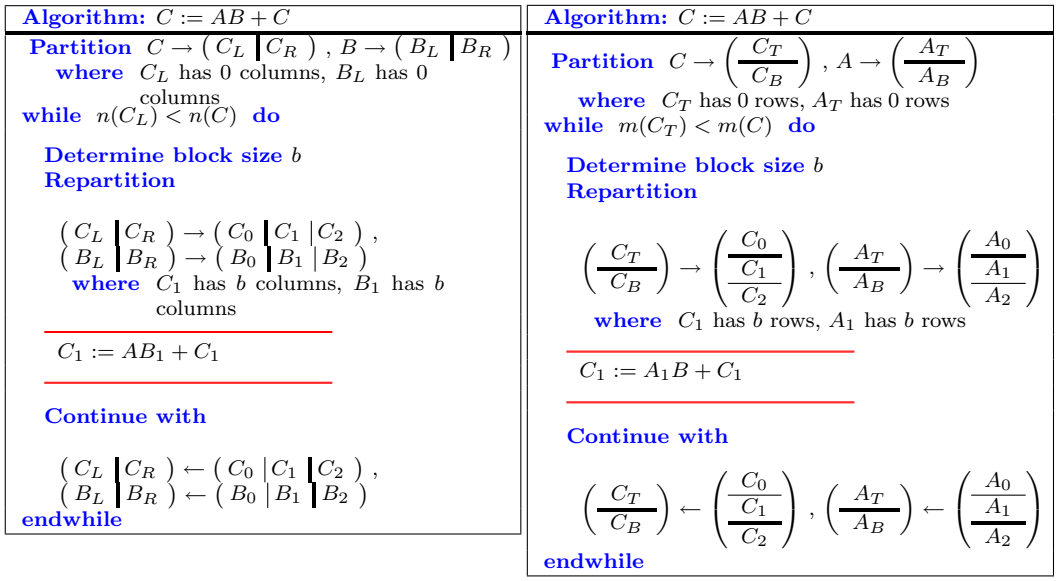

Figure 3.2: Blocked matrix-panel (left) and panel-matrix (right) algorithms for normal-normal **gemm**.

communicating B, and when $k \ll (m, n)$, one should avoid communicating C. The panel-panel blocking scheme is shown for normal-normal **gemm** in Fig. 3.1, and the matrix-panel and panel-matrix schemes are in Fig. 3.2. We have now defined 12 categories to investigate. It is important to recognize that algorithms very similar to our results are implemented in PLAPACK and discussed by Gunnels et al[15]. The main purpose of this chapter is to demonstrate the utility of the new notation for formally deriving parallel algorithms.

It should also be noted that the techniques presented for **gemm** directly apply to the other matrix-matrix multiplication routines in the Level 3 BLAS[7], such as `syrk`, `syr2k`, `symm`, and `trmm`.

3.1 Normal-Normal

Normal-normal **gemm** is described by

$$C := AB,$$

and if we choose to solve the problem in parallel with C ending up in a matrix distribution, then we can express each process's workload as

$$C^{\mathcal{C},\mathcal{R}} := [AB]^{\mathcal{C},\mathcal{R}}, \tag{3.2}$$

which is (well-defined) shorthand for

$$C^{\mathcal{C}^s,\mathcal{R}^t} := [AB]^{\mathcal{C}^s,\mathcal{R}^t}, \ \forall \ (s,t) \in \mathcal{T}_C \times \mathcal{T}_R.$$

Thus if each process performs the above operation, the operation $C = AB$ has been parallelized. Since the stationary C case is the simplest, we derive it first, followed by the stationary A and B algorithms, as they are conceptually the same in their approach.

3.1.1 Stationary C

Starting with the basic description of the parallel update, (3.2), we can trivially modify it as

$$C^{\mathcal{C},\mathcal{R}} := \left[A^{\mathbb{N},\mathbb{N}} B^{\mathbb{N},\mathbb{N}}\right]^{\mathcal{C},\mathcal{R}},$$

then apply the column filter to the columns of A, and the row filter to the rows of B, such that

$$C^{\mathcal{C},\mathcal{R}} := A^{\mathcal{C},\mathbb{N}} B^{\mathbb{N},\mathcal{R}}. \tag{3.3}$$

Our stationary C algorithm can now easily be described. Recognizing that we begin with A, B, and C in matrix distributions, then using relations (2.16) and (2.15), we can simply Allgather $A^{\mathcal{C},\mathcal{R}}$ across the process rows and $B^{\mathcal{C},\mathcal{R}}$ across the process columns to respectively form $A^{\mathcal{C},\mathbb{N}}$ and $B^{\mathbb{N},\mathcal{R}}$. The update is then completed by performing a local `gemm` on every process, shown in (3.3). The parallelization of a single iteration, within a panel-panel blocking scheme, is shown in Fig. 3.3. In this case, it is assumed that we begin with the column filter of A aligned with the column filter of C, and similarly for the row filters of B and C. If this is not the case, a call to Send-recv is needed for each misalignment.

$A_1^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$B_1^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_C}$
$C^{\mathcal{C},\mathcal{R}}$	$+=$	$A_1^{\mathcal{C},\mathbb{N}} B_1^{\mathbb{N},\mathcal{R}}$	`gemm`

Figure 3.3: Parallelization of a single iteration of normal-normal **gemm** with stationary C.

3.1.2 Stationary A/B

Because the derivations of the stationary A and B algorithms are so similar, we will derive the stationary A algorithm and then simply state the analogous stationary B result. For the stationary A case, we rely upon (2.5) to recognize that

$$[AB]^{\mathcal{C},\mathbb{N}} = \sum_{q \in \mathcal{T}_R} A^{\mathcal{C},\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}},$$

and after applying $[\,\cdot\,]^{\mathbb{N},\mathcal{R}}$ to both sides

$$\left[[AB]^{\mathcal{C},\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}} = \left[\sum_{q\in\mathcal{T}_R} A^{\mathcal{C},\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}}.$$

We now make use of Eq. (2.3) to show that $\left[[\,\cdot\,]^{\mathcal{C},\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}} = [\,\cdot\,]^{\mathcal{C},\mathcal{R}}$, and thus

$$[AB]^{\mathcal{C},\mathcal{R}} = \left[\sum_{q\in\mathcal{T}_R} A^{\mathcal{C},\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}}.$$

In expanded form we have that

$$[AB]^{\mathcal{C}^s,\mathcal{R}^t} = \left[\sum_{q\in\mathcal{T}_R} A^{\mathcal{C}^s,\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}^t}, \quad \forall (s,t) \in \mathcal{T}_C \times \mathcal{T}_R.$$

It is clear that the quantity in brackets on the right-hand side is independent of t and thus invariant within process rows. Thus if each process $(s,t) \in \mathcal{T}_C \times \mathcal{T}_R$ performs the local **gemm** $A^{\mathcal{C}^s,\mathcal{R}^t} B^{\mathcal{R}^t,\mathbb{N}}$, sums the result over its process row via a Reduce, and then applies $[\,\cdot\,]^{\mathbb{N},\mathcal{R}^t}$ to the result of the summation, then it has computed $[AB]^{\mathcal{C}^s,\mathcal{R}^t}$. While we have fully parallelized the local **gemm**, the algorithm is not yet optimal, as it implies that each process receives the full result of $\sum_{q\in\mathcal{T}_R} A^{\mathcal{C}^s,\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}}$ and then stores only the portion left after applying $[\,\cdot\,]^{\mathbb{N},\mathcal{R}^t}$. Clearly this involves unnecessary communication, and we recognize that the Reduce-scatter operation allows one to spread the result of a summation within the group of processes that performed the summation. Since $\cup_{q\in\mathcal{T}_R} \mathcal{R}^q = \mathbb{N}$ and $\mathcal{R}^i \cap \mathcal{R}^j = \emptyset, i \neq j$, we can use each process's row filter to select the columns it should receive from the Reduce-scatter. We now rewrite (3.2) as

$$C^{\mathcal{C},\mathcal{R}} := \sum_{q\in\mathcal{T}_R} \left[A^{\mathcal{C},\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}}, \tag{3.4}$$

or, in expanded form,

$$C^{\mathcal{C}^s,\mathcal{R}^t} := \sum_{q \in \mathcal{T}_R} \left[A^{\mathcal{C}^s,\mathcal{R}^q} B^{\mathcal{R}^q,\mathbb{N}} \right]^{\mathbb{N},\mathcal{R}^t}, \ \forall \ (s,t) \in \mathcal{T}_C \times \mathcal{T}_R.$$

Similarly, the stationary B algorithm is described by the equation

$$C^{\mathcal{C},\mathcal{R}} := \sum_{q \in \mathcal{T}_C} \left[A^{\mathbb{N},\mathcal{C}^q} B^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}}. \tag{3.5}$$

The parallelizations of a single iteration from both cases are shown in Fig. 3.4, with each algorithm using the previously described blocking schemes.

$B_1^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_R,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_C,\mathbb{N}}$	Send-recv
$B_1^{\mathcal{R},\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_R,\mathbb{N}}$	Allgather over $\widehat{\mathcal{T}_C}$
$\hat{X}^{\mathcal{C},\mathbb{N}}$	$:=$	$A^{\mathcal{C},\mathcal{R}} B_1^{\mathcal{R},\mathbb{N}}$	`gemm`
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$\sum_{q \in \mathcal{T}_R} \left[\hat{X}_q^{\mathcal{C},\mathbb{N}} \right]^{\mathbb{N},\mathcal{R}}$	Reduce-scatter over $\widehat{\mathcal{T}_R}$

$A_1^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$A_1^{\mathbb{N},\mathcal{M}_C}$	$\Leftarrow$	$A_1^{\mathbb{N},\mathcal{M}_R}$	Send-recv
$A_1^{\mathbb{N},\mathcal{C}}$	$\Leftarrow$	$A_1^{\mathbb{N},\mathcal{M}_C}$	Allgather over $\widehat{\mathcal{T}_R}$
$\hat{X}^{\mathbb{N},\mathcal{R}}$	$:=$	$A_1^{\mathbb{N},\mathcal{C}} B^{\mathcal{C},\mathcal{R}}$	`gemm`
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$\sum_{q \in \mathcal{T}_C} \left[\hat{X}_q^{\mathbb{N},\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$

Figure 3.4: Parallelizations of a single iteration of normal-normal **gemm** with stationary A (top) and B (bottom).

3.2 Normal-Transposed/Transposed-Normal

Normal-transposed and transposed-normal **gemm** are respectively described by

$$\begin{aligned} C &:= AB^T, \text{ and} \\ C &:= A^T B, \end{aligned}$$

while the elemental counterparts are

$$C^{\mathcal{C},\mathcal{R}} := \left[AB^T\right]^{\mathcal{C},\mathcal{R}}, \text{ and} \tag{3.6}$$

$$C^{\mathcal{C},\mathcal{R}} := \left[A^T B\right]^{\mathcal{C},\mathcal{R}}. \tag{3.7}$$

Making use of duality, for each analysis of a normal-transposed case we will list the corresponding transposed-normal result. We again start with the simplest case, the stationary C algorithm.

3.2.1 Stationary C

We derive the normal-transposed stationary C algorithm by first expanding $\left[AB^T\right]^{\mathcal{C},\mathcal{R}}$ as

$$\left[AB^T\right]^{\mathcal{C},\mathcal{R}} = A^{\mathcal{C},\mathbb{N}} \left[B^T\right]^{\mathbb{N},\mathcal{R}},$$

then applying (2.2) to change the order of filter application and transposition of B so that

$$\left[AB^T\right]^{\mathcal{C},\mathcal{R}} = A^{\mathcal{C},\mathbb{N}} \left[B^{\mathcal{R},\mathbb{N}}\right]^T.$$

The normal-transposed stationary C update is thus

$$C^{\mathcal{C},\mathcal{R}} := A^{\mathcal{C},\mathbb{N}} \left[B^{\mathcal{R},\mathbb{N}}\right]^T, \tag{3.8}$$

and the transposed-normal update is analogously

$$C^{\mathcal{C},\mathcal{R}} := \left[A^{\mathbb{N},\mathcal{C}}\right]^T B^{\mathbb{N},\mathcal{R}}. \tag{3.9}$$

A single iteration of each is shown in Fig. 3.5.

$A_1^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_R,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_C,\mathbb{N}}$	Send-recv
$B_1^{\mathcal{R},\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_R,\mathbb{N}}$	Allgather over $\widehat{\mathcal{T}_C}$
$C^{\mathcal{C},\mathcal{R}}$	$+=$	$A_1^{\mathcal{C},\mathbb{N}} \left[B_1^{\mathcal{R},\mathbb{N}}\right]^T$	`gemm`

$B_1^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_C}$
$A_1^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$A_1^{\mathbb{N},\mathcal{M}_C}$	$\Leftarrow$	$A_1^{\mathbb{N},\mathcal{M}_R}$	Send-recv
$A_1^{\mathbb{N},\mathcal{C}}$	$\Leftarrow$	$A_1^{\mathbb{N},\mathcal{M}_C}$	Allgather over $\widehat{\mathcal{T}_R}$
$C^{\mathcal{C},\mathcal{R}}$	$+=$	$\left[A_1^{\mathbb{N},\mathcal{C}}\right]^T B_1^{\mathbb{N},\mathcal{R}}$	`gemm`

Figure 3.5: Parallelizations of a single iteration of normal-transposed **gemm** with stationary C (top) and transposed-normal **gemm** with stationary C (bottom).

3.2.2 NT Stationary A, TN Stationary B

Our goal is to find an algorithm that performs the normal-transposed **gemm** update, (3.6), in parallel, but without communicating A. Using (2.5),

we can expand $\left[AB^T\right]^{\mathcal{C},\mathbb{N}}$ as

$$\left[AB^T\right]^{\mathcal{C},\mathbb{N}} = \sum_{q \in \mathcal{T}_R} A^{\mathcal{C},\mathcal{R}^q} \left[B^T\right]^{\mathcal{R}^q,\mathbb{N}},$$

and after applying (2.2) to switch the order of the transposition and filter application on B,

$$\left[AB^T\right]^{\mathcal{C},\mathbb{N}} = \sum_{q \in \mathcal{T}_R} A^{\mathcal{C},\mathcal{R}^q} \left[B^{\mathbb{N},\mathcal{R}^q}\right]^T.$$

Applying $[\,\cdot\,]^{\mathbb{N},\mathcal{R}}$ to both sides, it follows that

$$\left[AB^T\right]^{\mathcal{C},\mathcal{R}} = \left[\sum_{q \in \mathcal{T}_R} A^{\mathcal{C},\mathcal{R}^q} \left[B^{\mathbb{N},\mathcal{R}^q}\right]^T\right]^{\mathbb{N},\mathcal{R}}.$$

We can now recast (3.6) as the normal-transposed update with stationary A,

$$C^{\mathcal{C},\mathcal{R}} := \sum_{q \in \mathcal{T}_R} \left[A^{\mathcal{C},\mathcal{R}^q} \left[B^{\mathbb{N},\mathcal{R}^q}\right]^T\right]^{\mathbb{N},\mathcal{R}}, \tag{3.10}$$

and similarly, we can express (3.7) as the transposed-normal update with stationary B,

$$C^{\mathcal{C},\mathcal{R}} := \sum_{q \in \mathcal{T}_C} \left[\left[A^{\mathcal{C}^q,\mathbb{N}}\right]^T B^{\mathcal{C}^q,\mathcal{R}}\right]^{\mathcal{C},\mathbb{N}}. \tag{3.11}$$

The parallelizations of the matrix-panel normal-transposed and panel-matrix transposed-normal blocking schemes are shown in Fig. 3.6.

3.2.3 NT Stationary B, TN Stationary A

We now seek an algorithm for the normal-transposed **gemm** that avoids communicating the matrix B. One solution is to form the AB^T update in the

$B_1^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_C}$
$\hat{X}^{\mathcal{C},\mathbb{N}}$	$:=$	$A^{\mathcal{C},\mathcal{R}} \left[B_1^{\mathbb{N},\mathcal{R}}\right]^T$	`gemm`
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$\sum_{q\in\mathcal{T}_R} \left[\hat{X}_q^{\mathcal{C},\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}}$	Reduce-scatter over $\widehat{\mathcal{T}_R}$

$A_1^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\hat{X}^{\mathbb{N},\mathcal{R}}$	$:=$	$\left[A_1^{\mathcal{C},\mathbb{N}}\right]^T B^{\mathcal{C},\mathcal{R}}$	`gemm`
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$\sum_{q\in\mathcal{T}_C} \left[\hat{X}_q^{\mathbb{N},\mathcal{R}}\right]^{\mathcal{C},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$

Figure 3.6: Parallelizations of a single iteration of normal-transposed **gemm** with stationary A (top) and transposed-normal **gemm** with stationary B (bottom).

transposed matrix distribution, and then to redistribute it into the matrix distribution. Using (2.5) to expand $\left[AB^T\right]^{\mathbb{N},\mathcal{C}}$,

$$\left[AB^T\right]^{\mathbb{N},\mathcal{C}} = \sum_{q\in\mathcal{T}_R} A^{\mathbb{N},\mathcal{R}^q} \left[B^T\right]^{\mathcal{R}^q,\mathcal{C}},$$

and, using (2.2) to reverse the order of the transposition and filter application on B, it follows that

$$\left[AB^T\right]^{\mathbb{N},\mathcal{C}} = \sum_{q\in\mathcal{T}_R} A^{\mathbb{N},\mathcal{R}^q} \left[B^{\mathcal{C},\mathcal{R}^q}\right]^T.$$

Applying $[\,\cdot\,]^{\mathcal{R},\mathbb{N}}$ to both sides,

$$\left[AB^T\right]^{\mathcal{R},\mathcal{C}} = \left[\sum_{q\in\mathcal{T}_R} A^{\mathbb{N},\mathcal{R}^q} \left[B^{\mathcal{C},\mathcal{R}^q}\right]^T\right]^{\mathcal{R},\mathbb{N}}.$$

Now, we simply apply the definition of the distribution transposition operator, $[\,\cdot\,]^{\leftrightarrow}$, to find that

$$\left[AB^T\right]^{\mathcal{C},\mathcal{R}} = \left[\left[\sum_{q\in\mathcal{T}_R} A^{\mathbb{N},\mathcal{R}^q}\left[B^{\mathcal{C},\mathcal{R}^q}\right]^T\right]^{\mathcal{R},\mathbb{N}}\right]^{\leftrightarrow}.$$

We again apply the concept of a Reduce-scatter and write our normal-transposed update with stationary B as

$$C^{\mathcal{C},\mathcal{R}} := \left[\sum_{q\in\mathcal{T}_R}\left[A^{\mathbb{N},\mathcal{R}^q}\left[B^{\mathcal{C},\mathcal{R}^q}\right]^T\right]^{\mathcal{R},\mathbb{N}}\right]^{\leftrightarrow}, \tag{3.12}$$

and the dual update, the transposed-normal with stationary A case, as

$$C^{\mathcal{C},\mathcal{R}} := \left[\sum_{q\in\mathcal{T}_C}\left[\left[A^{\mathcal{C}^q,\mathcal{R}}\right]^T B^{\mathcal{C}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{C}}\right]^{\leftrightarrow}. \tag{3.13}$$

These expressions imply the parallelizations listed in Fig. 3.7.

3.3 Transposed-Transposed

The transposed-transposed **gemm** operation is

$$C := A^T B^T,$$

and the elemental version is

$$C^{\mathcal{C},\mathcal{R}} := \left[A^T B^T\right]^{\mathcal{C},\mathcal{R}}. \tag{3.14}$$

As with the normal-normal **gemm**, we begin with the simplest case, stationary C, then simultaneously describe the stationary A and stationary B algorithms.

$A_1^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_C}$
$\hat{X}^{\mathbb{N},\mathcal{C}}$	$:=$	$A_1^{\mathbb{N},\mathcal{R}} \left[B^{\mathcal{C},\mathcal{R}}\right]^T$	`gemm`
$X^{\mathcal{R},\mathcal{C}}$	$:=$	$\sum_{q\in\mathcal{T}_R} \left[\hat{X}_q^{\mathbb{N},\mathcal{C}}\right]^{\mathcal{R},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_R}$
$X^{\mathcal{C},\mathcal{R}}$	$\Leftarrow$	$X^{\mathcal{R},\mathcal{C}}$	Transpose the distribution
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$X^{\mathcal{C},\mathcal{R}}$	`axpy`s

$B_1^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\hat{X}^{\mathcal{R},\mathbb{N}}$	$:=$	$\left[A^{\mathcal{C},\mathcal{R}}\right]^T B_1^{\mathcal{C},\mathbb{N}}$	`gemm`
$X^{\mathcal{R},\mathcal{C}}$	$:=$	$\sum_{q\in\mathcal{T}_C} \left[\hat{X}_q^{\mathcal{R},\mathbb{N}}\right]^{\mathbb{N},\mathcal{C}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$
$X^{\mathcal{C},\mathcal{R}}$	$\Leftarrow$	$X^{\mathcal{R},\mathcal{C}}$	Transpose the distribution
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$X^{\mathcal{C},\mathcal{R}}$	`axpy`s

Figure 3.7: Parallelizations of a single iteration of normal-transposed **gemm** with stationary B (top) and transposed-normal **gemm** with stationary A (bottom).

3.3.1 Stationary C

We can quickly derive the stationary C algorithm by first expanding

$$\left[A^T B^T\right]^{\mathcal{C},\mathcal{R}} = \left[A^T\right]^{\mathcal{C},\mathbb{N}} \left[B^T\right]^{\mathbb{N},\mathcal{R}},$$

and then applying (2.2) to find that

$$\left[A^T B^T\right]^{\mathcal{C},\mathcal{R}} = \left[A^{\mathbb{N},\mathcal{C}}\right]^T \left[B^{\mathcal{R},\mathbb{N}}\right]^T.$$

Thus our transposed-transposed stationary C approach is

$$C^{\mathcal{C},\mathcal{R}} := \left[A^{\mathbb{N},\mathcal{C}}\right]^T \left[B^{\mathcal{R},\mathbb{N}}\right]^T, \tag{3.15}$$

and the blocked algorithm is shown in detail in Fig. 3.8.

$A_1^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$A_1^{\mathbb{N},\mathcal{M}_C}$	$\Leftarrow$	$A_1^{\mathbb{N},\mathcal{M}_R}$	Send-recv
$A_1^{\mathbb{N},\mathcal{C}}$	$\Leftarrow$	$A_1^{\mathbb{N},\mathcal{M}_C}$	Allgather over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_R,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_C,\mathbb{N}}$	Send-recv
$B_1^{\mathcal{R},\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_R,\mathbb{N}}$	Allgather over $\widehat{\mathcal{T}_C}$
$C^{\mathcal{C},\mathcal{R}}$	$+=$	$\left[A_1^{\mathbb{N},\mathcal{C}}\right]^T \left[B_1^{\mathcal{R},\mathbb{N}}\right]^T$	`gemm`

Figure 3.8: Parallelization of a single iteration of transposed-transposed **gemm** with stationary C.

3.3.2 Stationary A/B

The last two algorithms may be found by trivially modifying the results from the normal-transposed stationary B and transposed-normal stationary A algorithms, respectively. Thus the transposed-transposed stationary A and stationary B expressions are

$$C^{\mathcal{C},\mathcal{R}} := \left[\sum_{q\in\mathcal{T}_C} \left[\left[A^{\mathcal{C}^q,\mathcal{R}}\right]^T \left[B^{\mathbb{N},\mathcal{C}^q}\right]^T\right]^{\mathbb{N},\mathcal{C}}\right]^{\leftrightarrow}, \text{ and} \tag{3.16}$$

$$C^{\mathcal{C},\mathcal{R}} := \left[\sum_{q\in\mathcal{T}_R} \left[\left[A^{\mathcal{R}^q,\mathbb{N}}\right]^T \left[B^{\mathcal{C},\mathcal{R}^q}\right]^T\right]^{\mathcal{R},\mathbb{N}}\right]^{\leftrightarrow}, \tag{3.17}$$

and the procedure for a single iteration of each is described in Fig. 3.9.

$B_1^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$B_1^{\mathbb{N},\mathcal{M}_C}$	$\Leftarrow$	$B_1^{\mathbb{N},\mathcal{M}_R}$	Send-recv
$B_1^{\mathbb{N},\mathcal{C}}$	$\Leftarrow$	$B_1^{\mathbb{N},\mathcal{M}_C}$	Allgather over $\widehat{\mathcal{T}_R}$
$\hat{X}^{\mathcal{R},\mathbb{N}}$	$:=$	$\left[A^{\mathcal{C},\mathcal{R}}\right]^T \left[B_1^{\mathbb{N},\mathcal{C}}\right]^T$	`gemm`
$X^{\mathcal{R},\mathcal{C}}$	$:=$	$\sum_{q\in\mathcal{T}_C} \left[\hat{X}_q^{\mathcal{R},\mathbb{N}}\right]^{\mathbb{N},\mathcal{C}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$
$X^{\mathcal{C},\mathcal{R}}$	$\Leftarrow$	$X^{\mathcal{R},\mathcal{C}}$	Transpose the distribution
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$X^{\mathcal{C},\mathcal{R}}$	`axpy`s

$A_1^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$A_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$A_1^{\mathcal{M}_R,\mathbb{N}}$	$\Leftarrow$	$A_1^{\mathcal{M}_C,\mathbb{N}}$	Send-recv
$A_1^{\mathcal{R},\mathbb{N}}$	$\Leftarrow$	$A_1^{\mathcal{M}_R,\mathbb{N}}$	Allgather over $\widehat{\mathcal{T}_C}$
$\hat{X}^{\mathbb{N},\mathcal{C}}$	$:=$	$\left[A_1^{\mathcal{R},\mathbb{N}}\right]^T \left[B^{\mathcal{C},\mathcal{R}}\right]^T$	`gemm`
$X^{\mathcal{R},\mathcal{C}}$	$:=$	$\sum_{q\in\mathcal{T}_R} \left[\hat{X}_q^{\mathbb{N},\mathcal{C}}\right]^{\mathcal{R},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_R}$
$X^{\mathcal{C},\mathcal{R}}$	$\Leftarrow$	$X^{\mathcal{R},\mathcal{C}}$	Transpose the distribution
$C_1^{\mathcal{C},\mathcal{R}}$	$+=$	$X^{\mathcal{C},\mathcal{R}}$	`axpy`s

Figure 3.9: Parallelizations of a single iteration of transposed-transposed **gemm** with stationary A (top) and B (bottom).

3.4 Performance Results

All twelve algorithms were implemented using C++/MPI and run on small subsets of TACC's Lonestar, a 5840 core Linux cluster with 4 cores per node. Each algorithm was tested on 16 and 64 cores with an algorithmic

blocksize of 128, using square matrices ranging up to $35,000 \times 35,000$ in size. ScaLAPACK 1.8 performance, using a distribution blocksize of 128, serves as a baseline for performance.

In order to test scalability, the stationary C variants were benchmarked against ScaLAPACK on 1024 cores of Ranger, a 62,976 core Linux cluster hosted at TACC. The algorithmic blocksize for the elemental algorithms, and the distribution blocksize for ScaLAPACK, were both chosen to be 256. Additionally, all parallel runs used MVAPICH 1.0 and GotoBLAS 1.26.

While running tests on square matrices would seem to only exercise a special case, no pipelining or double-buffering is employed, so the performance of each iteration of a particular algorithm does not noticeably change during execution. Lower bounds for matrix-panel, panel-matrix, and panel-panel multiplies are then respectively provided by the stationary A, B, and C algorithms.

Figs. 3.10, 3.11, 3.12, and 3.13 respectively show the performance results for the normal-normal, normal-transposed, transposed-normal, and transposed-transposed cases on Lonestar, where theoretical peak performance is set as the ceiling of the plots. While we have defined flops as floating-point operations, we define floating-point operations *per second* as FLOPS and report performance in gigaflops per second, or GFLOPS.

The performance of the stationary C algorithms is shown to be significantly higher than all other options for all four cases. Additionally, it is nearly

constant for all four cases, asymptoting at over 84% of theoretical peak on 16 cores, and over 75% of theoretical peak on 64 cores. All three parallel variants dominate ScaLAPACK performance, with the exception of ScaLAPACK reaching the performance of the stationary B case of transposed-transposed multiplication on 16 cores.

Fig. 3.14 shows the performance results on Ranger. In all four cases, the elemental versions greatly outperform ScaLAPACK. Due to the author having finite cycles on the machine, the matrix sizes were not increased until performance became asymptotic. Similar to the results on Lonestar, the performance of the C variant does not significantly change over the four transpose options.

While the performance results are clearly promising, the large gain in performance in simple normal-normal panel-panel multiplication suggests that significant portions of the overall performance gains are due to poor implementation of the ScaLAPACK algorithms rather than benefits of the presented theory.

Additionally, the back-transformation of the eigenvectors of the tridiagonal EVP using WY transforms requires $2n^3 + \mathcal{O}(n^2)$ flops. Because it is rich in parallel triangular matrix-matrix multiplication, translating the presented performance gains for general matrix-matrix multiplication would clearly be beneficial and is left for future work.

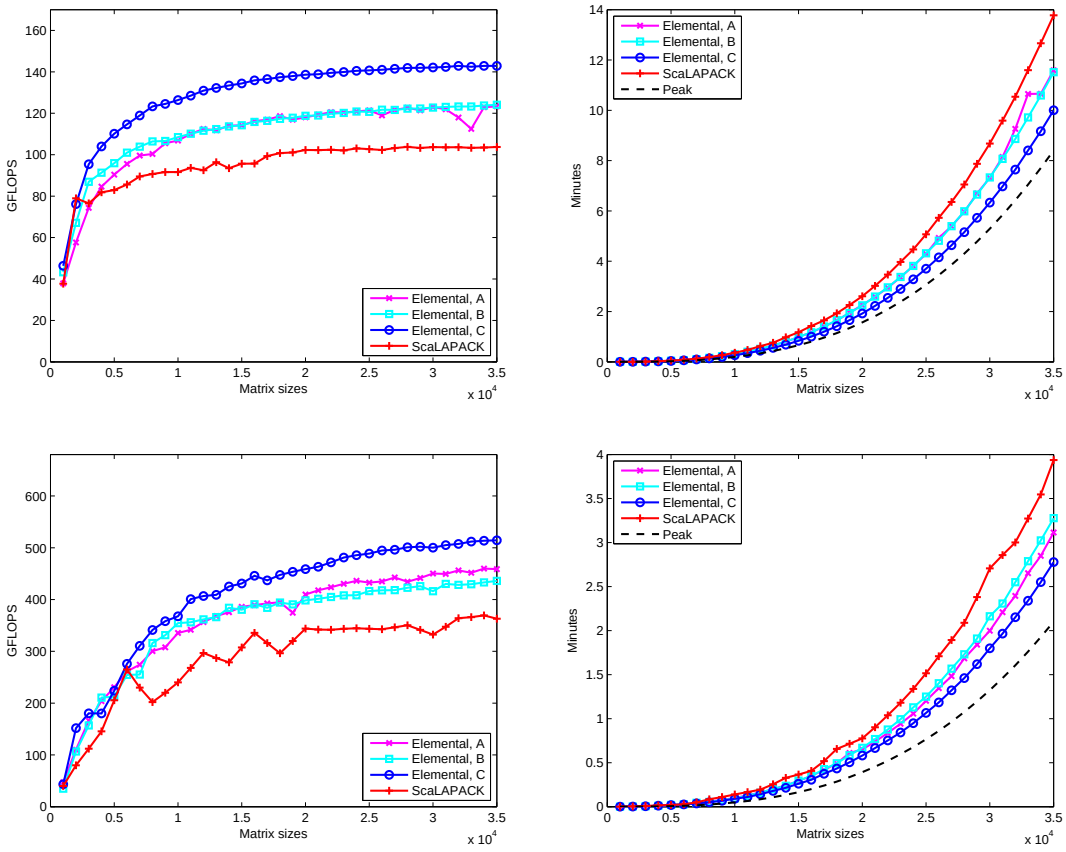

Figure 3.10: Performance and wall-clock time of parallel normal-normal **gemm** on 16 (top) and 64 (bottom) cores of Lonestar.

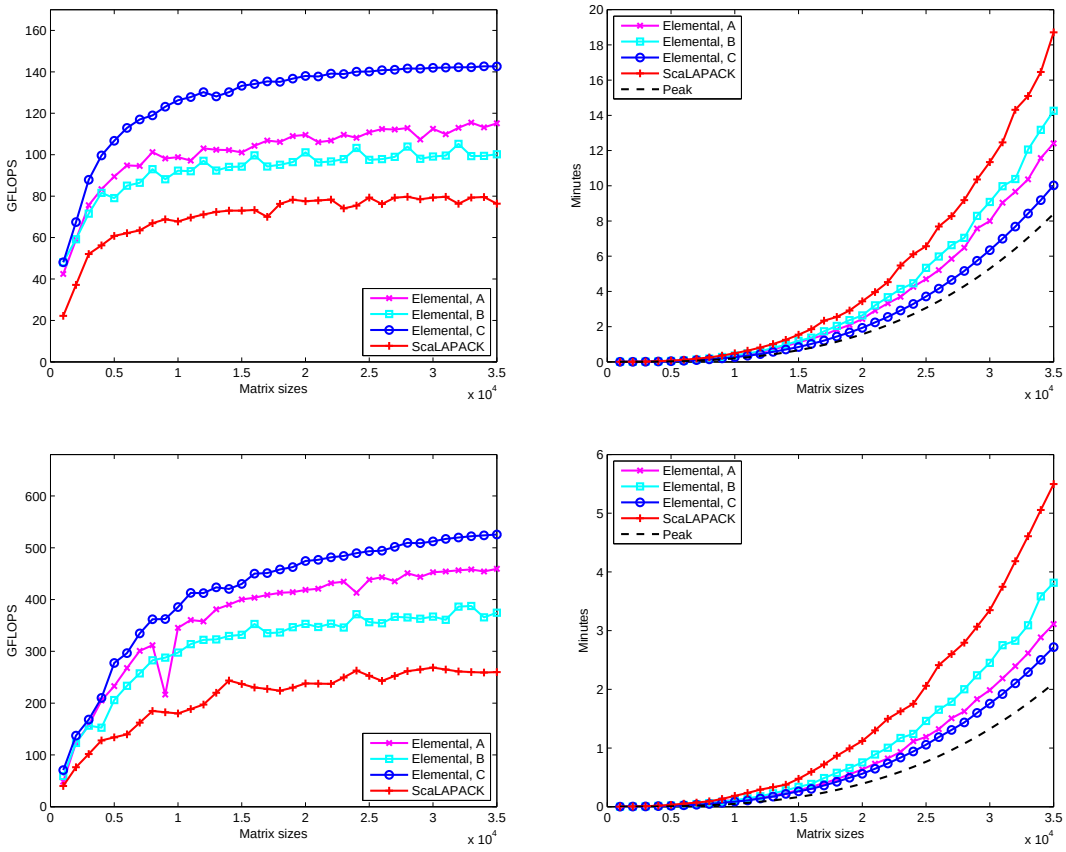

Figure 3.11: Performance and wall-clock time of parallel normal-transpose **gemm** on 16 (top) and 64 (bottom) cores of Lonestar.

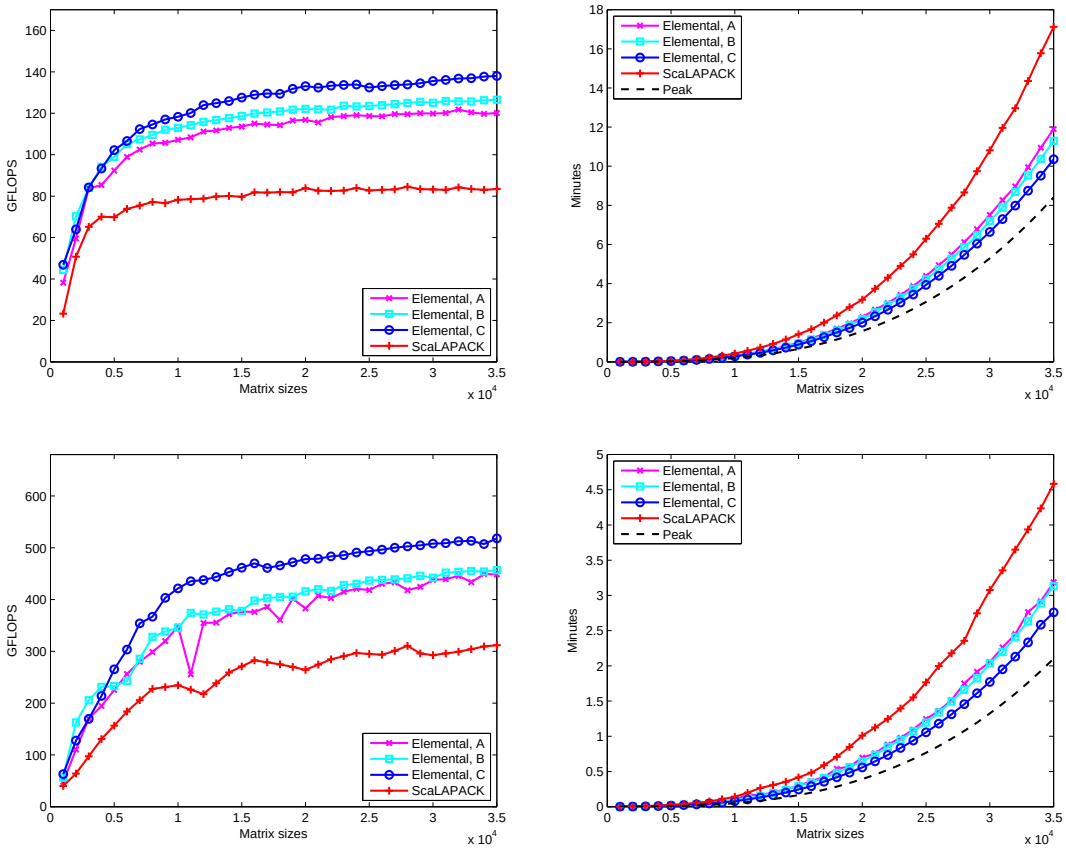

Figure 3.12: Performance and wall-clock time of parallel transposed-normal **gemm** on 16 (top) and 64 (bottom) cores of Lonestar.

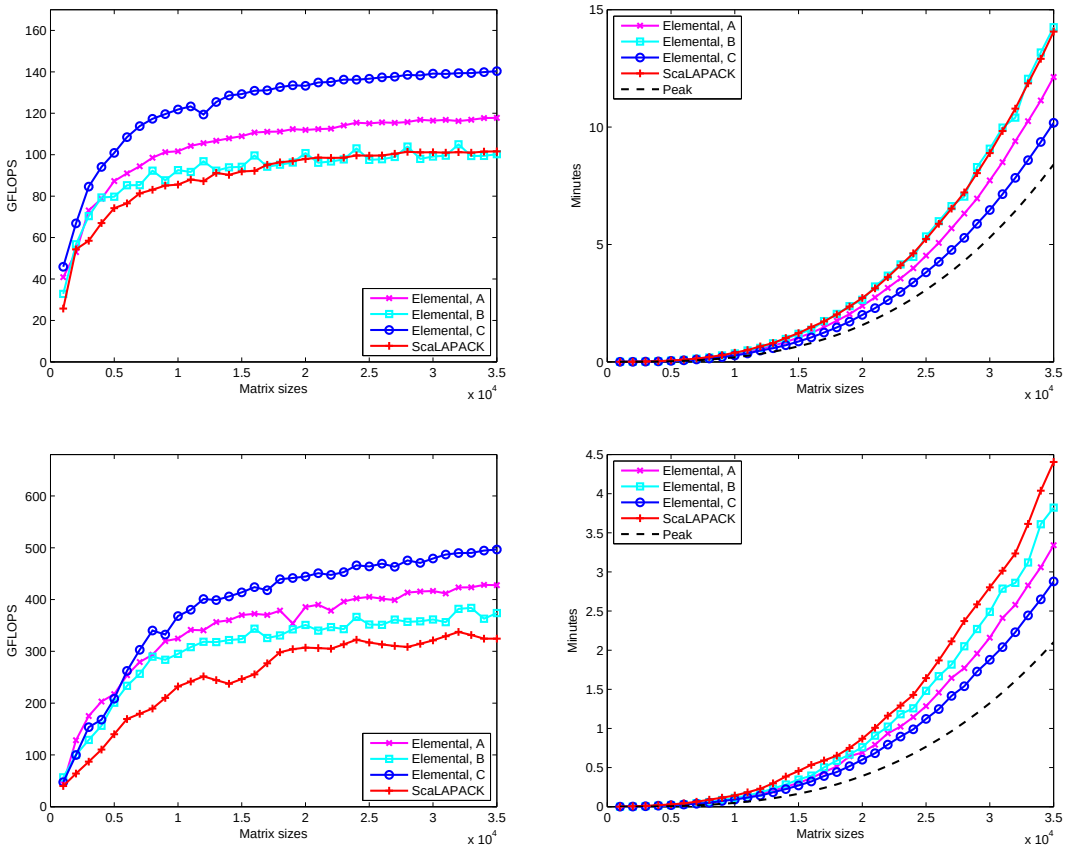

Figure 3.13: Performance and wall-clock time of parallel transposed-transposed **gemm** on 16 (top) and 64 (bottom) cores of Lonestar.

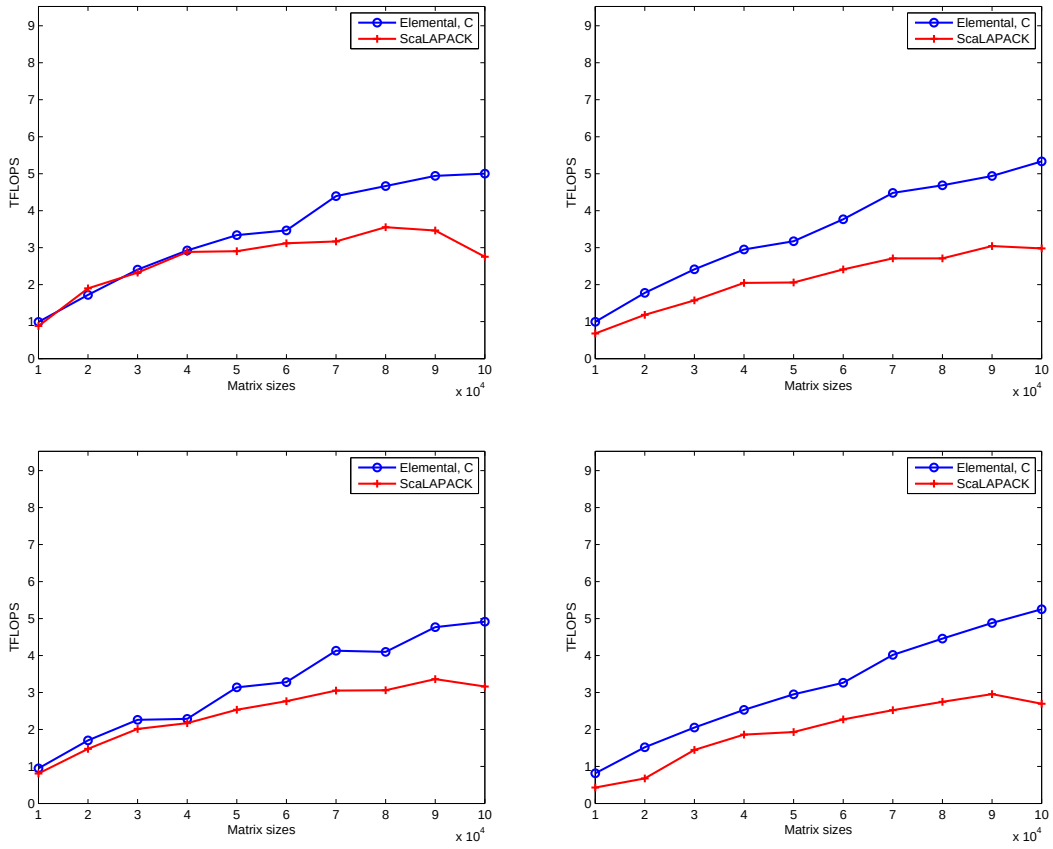

Figure 3.14: Performance of normal-normal (top-left), normal-transposed (top-right), transposed-normal (bottom-left), and transposed-transposed parallel **gemm** on 1024 cores of Ranger.

Chapter 4

Triangular Solves with Many Right-Hand Sides

The second operation we explore is the triangular solve with *many* right-hand sides. Because algorithms for solving against upper triangular matrices can be trivially modified for lower triangular matrices, we choose to analyze only the upper cases. The **trsm** operation is characterized by the solution, X, to either

$$\text{op}(U)\ X \;=\; \alpha B, \text{ or} \tag{4.1}$$

$$X\ \text{op}(U) \;=\; \alpha B, \tag{4.2}$$

where B is overwritten with X during the routine, and $\text{op}(\cdot)$ again returns either the input matrix or its transpose. The corresponding updates are respectively

$$B \;:=\; \alpha\ \text{op}(U)^{-1} B, \text{ and} \tag{4.3}$$

$$B \;:=\; \alpha B\ \text{op}(U)^{-1}. \tag{4.4}$$

The first equation represents the two *left* cases, since the triangular matrix is on the left, and thus the second equation represents the two *right* cases. The distinction from triangular solves with few right-hand sides is made because a

different approach is warranted, and in general, solves with few right-hand sides are not scalable. "Many" right-hand sides is chosen to be pn_b or more, where n_b is the algorithmic blocksize that yields optimal local **gemm** performance, and p is the number of processes to distribute the operation between.

The general approach for each iteration of the four algorithms is to perform a **trsm** with an $n_b \times n_b$ block of the triangular matrix, and then update a panel of B with a **gemm**. This is accomplished by putting the "right-hand sides", which are on the left for the *right* triangular solves, into a multivector distribution such that each right-hand side is owned by a single process. A trivially parallelized **trsm** update is then performed, followed by a **gemm** update. Each process is then left with n_b or more right-hand sides in their local **trsm** calls. While any multivector distribution can be used for the **trsm** update, it is beneficial to choose one that can be reused in the **gemm** update. A more thorough treatment for each of the four cases follows, where $\alpha = 1$ for simplicity.

4.1 Left-Upper-Normal

The left-upper-normal option of **trsm** solves

$$UX = B$$

by overwriting B with X, and thus it performs the update

$$B := U^{-1}B. \tag{4.5}$$

If we conformally partition

$$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right), X \rightarrow \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right), B \rightarrow \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right),$$

then

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) = \left(\begin{array}{c} U_{TL}X_T + U_{TR}X_B \\ \hline U_{BR}X_B \end{array}\right) = \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right). \tag{4.6}$$

A blocked recursive algorithm follows and is given in Fig. 4.1.

Algorithm: $[B] := U^{-1}B$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $B \rightarrow \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right)$

where U_{BR} is 0×0, B_B has 0 rows

while $m(U_{BR}) < m(U)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline B_1 \\ \hline B_2 \end{array}\right)$$

where U_{11} is $b \times b$, B_1 has b rows

$B_1 := U_{11}^{-1}B_1$

$B_0 := B_0 - U_{01}B_1$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline B_1 \\ \hline B_2 \end{array}\right)$$

endwhile

Figure 4.1: Blocked algorithm for performing multiple triangular solves when the upper-triangular matrix is on the LHS.

As previously mentioned, a matrix distribution is not ideal for the up-

date

$$B_1 := U_{11}^{-1} B_1,$$

as a row-wise multivector distribution of B_1 yields trivial parallelism. In order to determine *which* multivector distribution, we need to look at the other update,

$$B_0 := B_0 - U_{01} B_1.$$

Expressed in a matrix distribution,

$$B_0^{\mathcal{C},\mathcal{R}} := B_0^{\mathcal{C},\mathcal{R}} - [U_{01} B_1]^{\mathcal{C},\mathcal{R}}.$$

Because this is a panel-panel **gemm**, our previous analysis suggests the form

$$B_0^{\mathcal{C},\mathcal{R}} := B_0^{\mathcal{C},\mathcal{R}} - U_{01}^{\mathcal{C},\mathbb{N}} B_1^{\mathbb{N},\mathcal{R}}.$$

Using (2.15) and (2.28), with appropriate alignments $B_1^{\mathbb{N},\mathcal{R}}$ can be reached from $B_1^{\mathbb{N},\mathcal{M}_R}$ through an Allgather over $\widehat{\mathcal{T}_C}$, and $B_1^{\mathcal{C},\mathcal{R}}$ is a submatrix of $B_1^{\mathbb{N},\mathcal{R}}$. Then if B_1 is in a row-major row-wise multivector distribution for the local **trsm**, an intermediate redistribution into a matrix distribution is avoided by a direct movement into a row-projected row-wise partial matrix distribution. The parallelization of a single iteration of the blocked algorithm is shown in Fig. 4.2, where the underlined steps store the final results of partitions of B.

$U_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$U_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$B_1^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$B_1^{\mathbb{N},\mathcal{M}_R}$	$:=$	$\left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-1} B_1^{\mathbb{N},\mathcal{M}_R}$	`trsm`
$B_1^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$B_1^{\mathbb{N},\mathcal{M}_R}$	Allgather over $\widehat{\mathcal{T}_C}$
$\underline{B_1^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$B_1^{\mathbb{N},\mathcal{R}}$	Filtered copy
$U_{01}^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$U_{01}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\underline{B_0^{\mathcal{C},\mathcal{R}}}$	$-=$	$U_{01}^{\mathcal{C},\mathbb{N}} B_1^{\mathbb{N},\mathcal{R}}$	`gemm`

Figure 4.2: Parallelization of a single iteration of the left-upper-normal **trsm**.

4.2 Left-Upper-Transposed

The analysis of the left-upper-transposed case mirrors that of the previous case. We solve the system

$$U^T X = B$$

by overwriting B with X as

$$B := U^{-T} B. \tag{4.7}$$

If we partition our system of equations as

$$\left(\begin{array}{c|c} U_{TL}^T & 0 \\ \hline U_{TR}^T & U_{BR}^T \end{array}\right) \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) = \left(\begin{array}{c} U_{TL}^T X_T \\ \hline U_{TR}^T X_T + U_{BR}^T X_B \end{array}\right) = \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right), \tag{4.8}$$

then the blocked algorithm in Fig. 4.3 is easily identified. Just as before, the first update,

$$B_1 := U_{11}^{-T} B_1,$$

should be expressed with B_1 as a multivector instead of in a matrix distribution. The second update is

$$B_2 := B_2 - U_{12}^T B_1,$$

which is a panel-panel **gemm**, and is efficiently parallelized as

$$B_2^{\mathcal{C},\mathcal{R}} := B_2^{\mathcal{C},\mathcal{R}} - \left[U_{12}^{\mathbb{N},\mathcal{C}}\right]^T B_1^{\mathbb{N},\mathcal{R}}.$$

We again apply (2.15) and (2.28) to determine that B_1 should be a row-major row-wise multivector for the first update. A single iteration of the blocked parallel algorithm is shown in Fig. 4.4.

Algorithm: $[B] := U^{-T}B$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $B \rightarrow \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right)$

where U_{TL} is 0×0, B_T has 0 rows

while $m(U_{TL}) < m(U)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline B_1 \\ \hline B_2 \end{array}\right)$$

where U_{11} is $b \times b$, B_1 has b rows

$B_1 := U_{11}^{-T} B_1$

$B_2 := B_2 - U_{12}^T B_1$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline B_1 \\ \hline B_2 \end{array}\right)$$

endwhile

Figure 4.3: Blocked algorithm for performing multiple triangular solves when the upper-triangular matrix is transposed on the LHS.

4.3 Right-Upper-Normal

The right-upper-normal **trsm** operation solves the system

$$XU = B$$

by overwriting B with X as

$$B := BU^{-1}. \tag{4.9}$$

Partitioning

$$X \rightarrow \left(\begin{array}{c|c} X_L & X_R \end{array}\right), U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right), B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array}\right),$$

$U_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$U_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$B_1^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$B_1^{\mathbb{N},\mathcal{M}_R}$	$:=$	$\left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-T} B_1^{\mathbb{N},\mathcal{M}_R}$	`trsm`
$B_1^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$B_1^{\mathbb{N},\mathcal{M}_R}$	Allgather over $\widehat{\mathcal{T}_C}$
$\underline{B_1^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$B_1^{\mathbb{N},\mathcal{R}}$	Filtered copy
$U_{12}^{\mathbb{N},\mathcal{C}}$	$\Leftarrow$	$U_{12}^{\mathcal{C},\mathcal{R}}$	All-to-all $\widehat{\mathcal{T}_C}$, Send-recv, Allgather $\widehat{\mathcal{T}_R}$
$\underline{B_2^{\mathcal{C},\mathcal{R}}}$	$-=$	$\left[U_{12}^{\mathbb{N},\mathcal{C}}\right]^{T} B_1^{\mathbb{N},\mathcal{R}}$	`gemm`

Figure 4.4: Parallelization of a single iteration of the left-upper-transposed **trsm**.

then

$$\left(\begin{array}{c|c} X_L & X_R \end{array} \right) \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array} \right) = \left(\begin{array}{c|c} X_L U_{TL} & X_L U_{TR} + X_R U_{BR} \end{array} \right) = \left(\begin{array}{c|c} B_L & B_R \end{array} \right). \tag{4.10}$$

The solution method is shown in Fig. 4.5, and it is important to notice that in the first update,

$$B_1 := B_1 U_{11}^{-1},$$

the rows of B_1 can be solved for independently. It is then advantageous to put B_1 into a *column-wise* multivector. Analyzing the other update,

$$B_2 := B_2 - B_1 U_{12},$$

which is a panel-panel **gemm**, it is advantageous to distribute B_1 as $B_1^{\mathcal{M}_C,\mathbb{N}}$ for the first update so that we can cheaply gather $B_1^{\mathcal{C},\mathbb{N}}$ and perform

$$B_2^{\mathcal{C},\mathcal{R}} := B_2^{\mathcal{C},\mathcal{R}} - B_1^{\mathcal{C},\mathbb{N}} U_{12}^{\mathbb{N},\mathcal{R}}.$$

Algorithm: $[B] := BU^{-1}$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array}\right)$

where U_{TL} is 0×0, B_L has 0 columns

while $m(U_{TL}) < m(U)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c|c} B_L & B_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array}\right)$$

where U_{11} is $b \times b$, B_1 has b columns

$B_1 := B_1 U_{11}^{-1}$

$B_2 := B_2 - B_1 U_{12}$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c|c} B_L & B_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array}\right)$$

endwhile

Figure 4.5: Blocked algorithm for performing multiple triangular solves when the upper-triangular matrix is on the RHS.

The parallel scheme is shown in Fig. 4.6.

4.4 Right-Upper-Transposed

Going through the motions for the last case, we solve the system of equations

$$XU^T = B$$

$U_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$U_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$B_1^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_C,\mathbb{N}}$	$:=$	$B_1^{\mathcal{M}_C,\mathbb{N}} \left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-1}$	`trsm`
$B_1^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_C,\mathbb{N}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\underline{B_1^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathbb{N}}$	Filtered copy
$U_{12}^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$U_{12}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_C}$
$\underline{B_2^{\mathcal{C},\mathcal{R}}}$	$-=$	$B_1^{\mathcal{C},\mathbb{N}} U_{12}^{\mathbb{N},\mathcal{R}}$	`gemm`

Figure 4.6: Parallelization of a single iteration of the right-upper-normal **trsm**.

via the update

$$B := BU^{-T}. \tag{4.11}$$

Our serial algorithm, shown in Fig. 4.7, is found by partitioning the system of equations into the form

$$\left(\begin{array}{c|c} X_L & X_R \end{array} \right) \left(\begin{array}{c|c} U_{TL}^T & 0 \\ \hline U_{TR}^T & U_{BR}^T \end{array} \right) = \left(\begin{array}{c|c} X_L U_{TL}^T + X_R U_{TR}^T & X_R U_{BR}^T \end{array} \right) = \left(\begin{array}{c|c} B_L & B_R \end{array} \right), \tag{4.12}$$

and the corresponding parallel algorithm is demonstrated in Fig. 4.8.

4.5 Performance Results

The four **trsm** algorithms were implemented in the same style as the twelve **gemm** algorithms and were again benchmarked against ScaLAPACK. Figs. 4.9, 4.10, 4.11, and 4.12 respectively show the performance and wall-clock time of the left-upper-normal, left-upper-transposed, right-upper-normal, and

Algorithm: $[B] := BU^{-T}$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array}\right)$

where U_{BR} is 0×0, B_R has 0 columns

while $m(U_{BR}) < m(U)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c|c} B_L & B_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array}\right)$$

where U_{11} is $b \times b$, B_1 has b columns

$B_1 := B_1 U_{11}^{-T}$

$B_0 := B_0 - B_1 U_{01}^T$

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right), \left(\begin{array}{c|c} B_L & B_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} B_0 & B_1 & B_2 \end{array}\right)$$

endwhile

Figure 4.7: Blocked algorithm for performing multiple triangular solves when the upper-triangular matrix is transposed on the RHS.

right-upper-transposed cases on both 16 and 64 cores of Lonestar. Just as before, the ceilings of the performance plots are the theoretical peak performance of the given set of cores.

In all tests, the elemental implementation significantly outperforms ScaLAPACK. However, the margin shrinks greatly for the right-upper-normal case, where only $\sim$18% performance is gained at the maximum size tested on 64 cores.

$U_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$U_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$B_1^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$B_1^{\mathcal{M}_C,\mathbb{N}}$	$:=$	$B_1^{\mathcal{M}_C,\mathbb{N}} \left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-T}$	`trsm`
$B_1^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$B_1^{\mathcal{M}_C,\mathbb{N}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\underline{B_1^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$B_1^{\mathcal{C},\mathbb{N}}$	Filtered copy
$U_{01}^{\mathcal{R},\mathbb{N}}$	$\Leftarrow$	$U_{01}^{\mathcal{C},\mathcal{R}}$	All-to-all $\widehat{\mathcal{T}_R}$, Send-recv, Allgather $\widehat{\mathcal{T}_C}$
$\underline{B_0^{\mathcal{C},\mathcal{R}}}$	$-=$	$B_1^{\mathcal{C},\mathbb{N}} \left[U_{01}^{\mathcal{R},\mathbb{N}}\right]^{T}$	`gemm`

Figure 4.8: Parallelization of a single iteration of the right-upper-transposed **trsm**.

The back-transformation of the eigenvectors from the standard eigenvalue problem is done by solving for Φ in (1.5),

$$\Phi_A = U\Phi,$$

where $(\Phi_A, U) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n}$. This maps to a left-upper-normal **trsm** costing roughly n^3 flops, whose performance on 16 cores has been shown to increase nearly 40% over ScaLAPACK when $n > 15,000$.

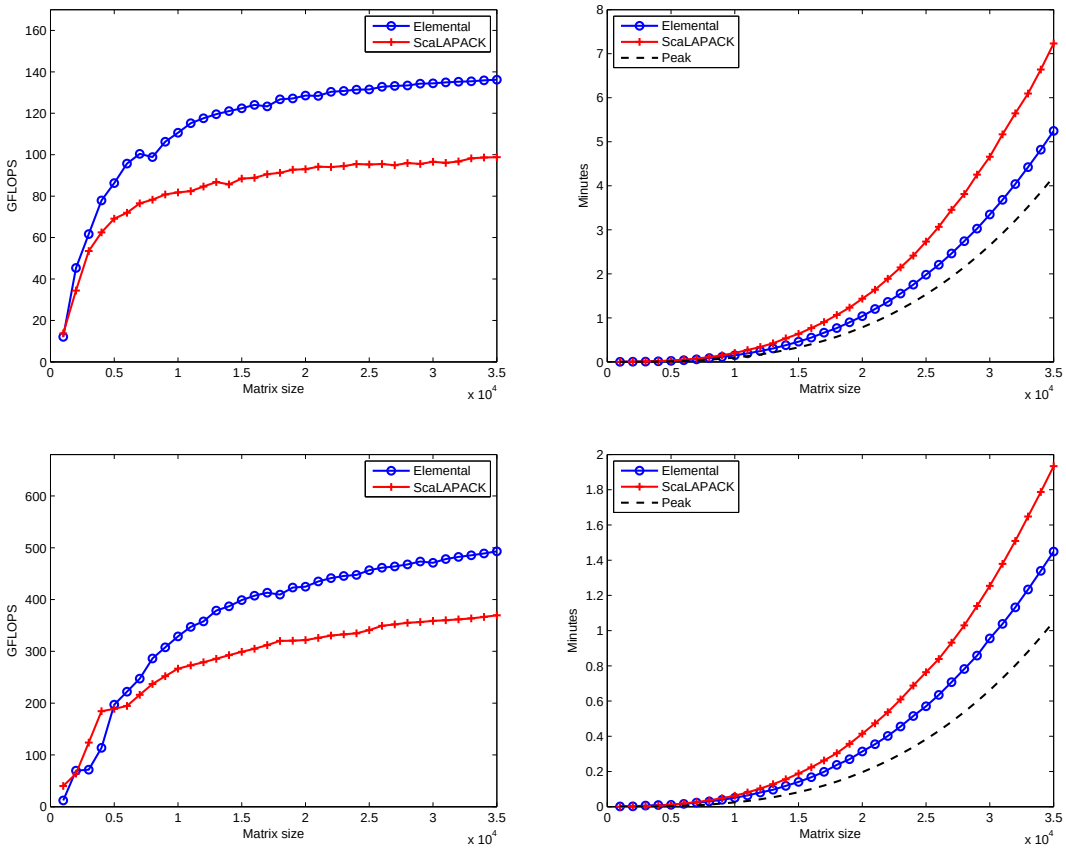

Figure 4.9: Performance and wall-clock time of left-upper-normal triangular solve with multiple RHS on 16 (top) and 64 (bottom) cores of Lonestar.

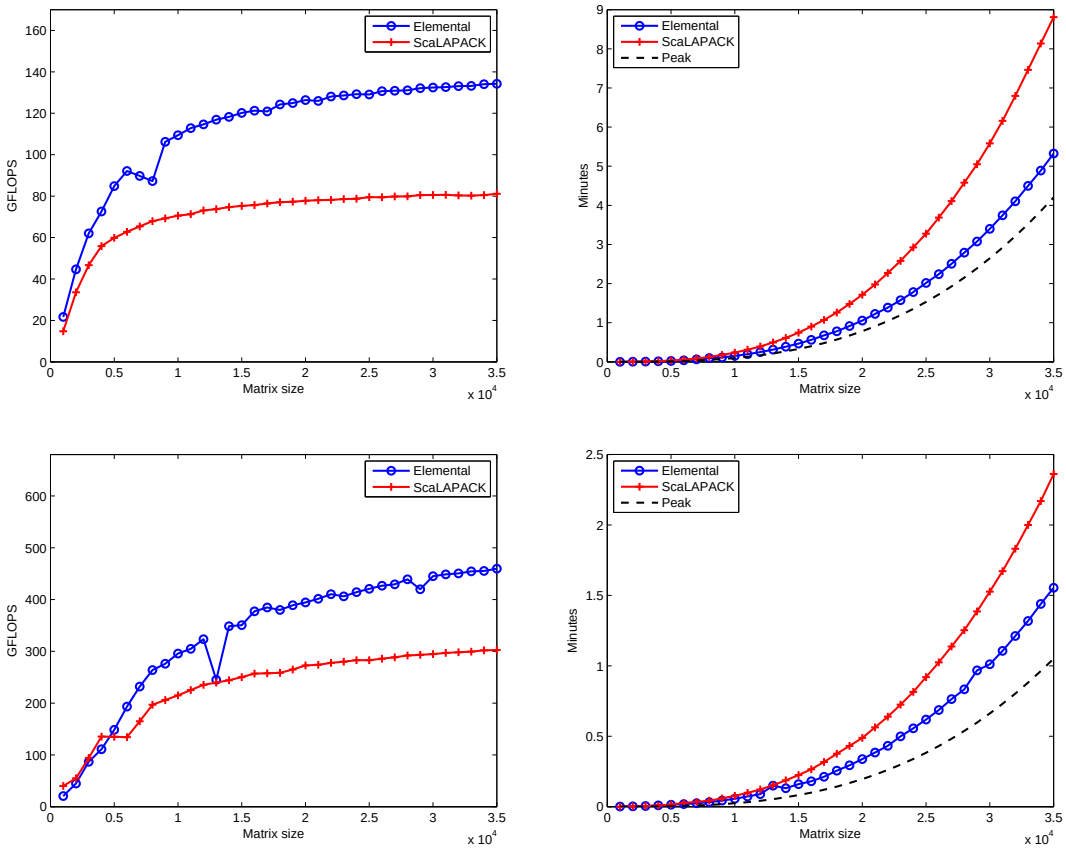

Figure 4.10: Performance and wall-clock time of left-upper-transposed triangular solve with multiple RHS on 16 (top) and 64 (bottom) cores of Lonestar.

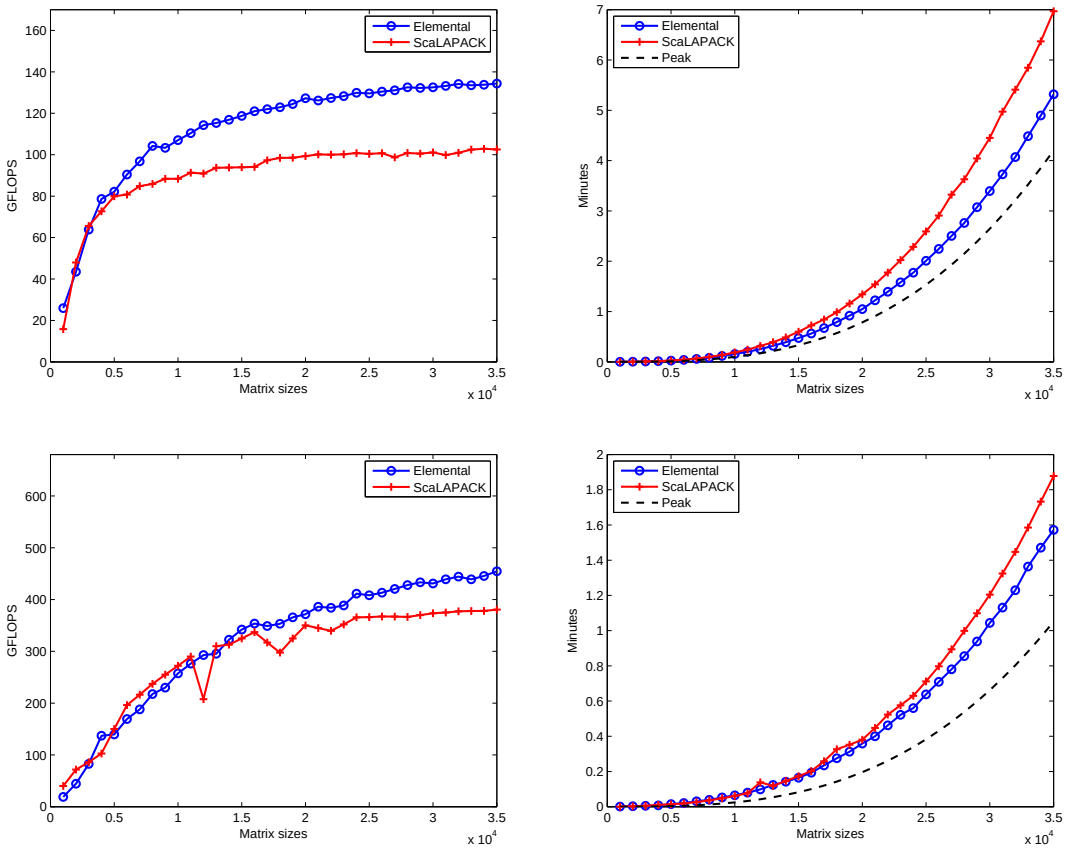

Figure 4.11: Performance and wall-clock time of right-upper-normal triangular solve with multiple RHS on 16 (top) and 64 (bottom) cores of Lonestar.

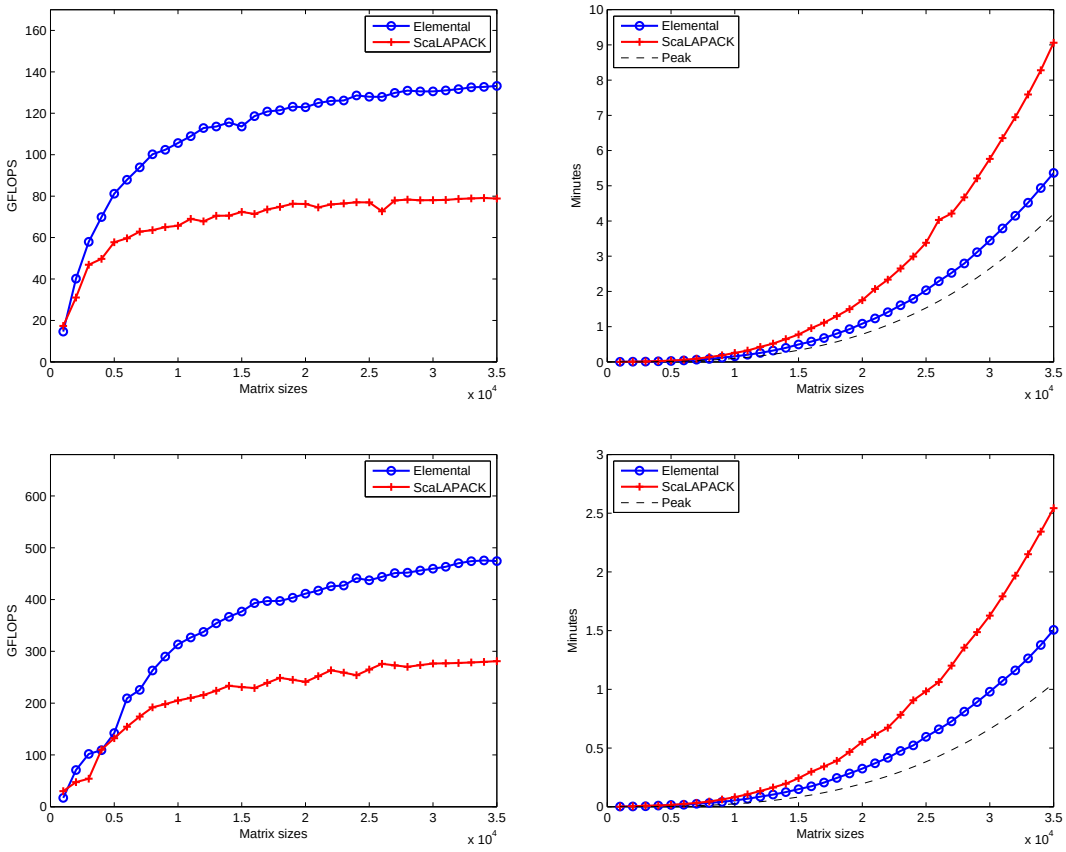

Figure 4.12: Performance and wall-clock time of right-upper-transposed triangular solve with multiple RHS on 16 (top) and 64 (bottom) cores of Lonestar.

Chapter 5

Cholesky Factorizations

Cholesky factorizations are a staple of dense linear algebra for both theoretical and computational purposes, and at $\frac{1}{3}n^3 + \mathcal{O}(n^2)$ flops[12], they are only half the price of LU factorizations. As previously discussed, the upper Cholesky factor, U, of our SPD mass matrix, M, will be used to transform the generalized EVP into standard form. We choose U to be upper, and thus it is defined to satisfy the relation

$$U^T U = M. \tag{5.1}$$

We also use the notation

$$U = \Gamma(M),$$

where $\Gamma(\cdot)$ returns the Cholesky factor of the input matrix. Because we would like to achieve the highest performance possible, three algorithmic variants will be derived and analyzed.

5.1 The Three Variants

Using Hoare triples and loop-invariants[18], formal proofs of correctness for implementations of algorithms can be systematically constructed[4]. A full

listing of the three Cholesky loop-invariants and algorithms is included to serve as a foundation for the similar work done for the reduction of the generalized EVP to standard form.

Starting with $M = U^T U$ and partitioning into a 2×2 matrix of blocks, we find that

$$\begin{aligned}\left(\begin{array}{c|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) &= \left(\begin{array}{c|c} U_{TL}^T & 0 \\ \hline U_{TR}^T & U_{BR}^T \end{array}\right)\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \\ &= \left(\begin{array}{c|c} U_{TL}^T U_{TL} & U_{TL}^T U_{TR} \\ \hline \star & U_{TR}^T U_{TR} + U_{BR}^T U_{BR} \end{array}\right),\end{aligned}$$

where a '$\star$' is used to denote a portion of a symmetric matrix that is not accessed. At the end of the factorization, the upper-triangle of M will be overwritten with U, and thus we will use $\hat{M}$ to denote the original state of M. We define the goal of the Cholesky operation by the Partitioned Matrix Expression (PME)

$$\left(\begin{array}{c|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) = \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline \star & U_{BR} \end{array}\right), \text{where } \left\{\begin{array}{l} \hat{M}_{TL} = U_{TL}^T U_{TL} \\ \hat{M}_{TR} = U_{TL}^T U_{TR} \\ \hat{M}_{BR} - U_{TR}^T U_{TR} = U_{BR}^T U_{BR}. \end{array}\right.$$

At this step in the derivation of loop-invariants, it is not always obvious how to group the update operations. For Cholesky factorizations, they are easily chosen and a simple Directed Acyclic Graph (DAG) of their dependencies is shown in Fig. 5.1. Because loop-invariants are restricted to legal intermediate states of an execution of the DAG, at least one, but not all, of the update operations must have been performed. Due to the operation dependencies and the mentioned requirement, there are only three loop-invariants. The first

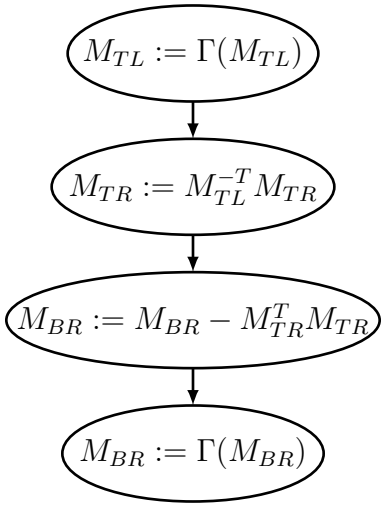

Figure 5.1: Update dependencies for a Cholesky factorization.

Invariant 1	$\left(\begin{array}{c\|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) = \left(\begin{array}{c\|c} U_{TL} & \hat{M}_{TL} \\ \hline \star & \hat{M}_{BR} \end{array}\right),\ \hat{M}_{TL} = U_{TL}^T U_{TL}$
Invariant 2	$\left(\begin{array}{c\|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) = \left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline \star & \hat{M}_{BR} \end{array}\right),\ \begin{array}{l} \hat{M}_{TL} = U_{TL}^T U_{TL} \\ \hat{M}_{TR} = U_{TL}^T U_{TR} \end{array}$
Invariant 3	$\left(\begin{array}{c\|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) = \left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline \star & \hat{M}_{BR} - U_{TR}^T U_{TR} \end{array}\right),\ \begin{array}{l} \hat{M}_{TL} = U_{TL}^T U_{TL} \\ \hat{M}_{TR} = U_{TL}^T U_{TR} \end{array}$

Figure 5.2: Loop-invariants for a Cholesky factorization.

invariant includes only the first update, the second invariant includes the first and second updates, and the third invariant includes the first, second, and third updates. The invariants are summarized in Fig. 5.2 and their corresponding algorithms may be found by assuming that the loop invariant holds at the beginning of a loop and finding the algebraic steps required for it to again hold after the repartitioning cycle[4]. The full algorithms are displayed in Fig. 5.3. It is immediately apparent that Variant 1 will not scale well, since the

update

$$M_{01} := M_{00}^{-T} M_{01}$$

corresponds to a triangular solve with the number of right-hand sides equal to the algorithmic blocksize. For this reason, we will only parallelize algorithmic Variants 2 and 3.

5.2 Parallelizing Variant 3

The analysis for parallelizing the Variant 3 algorithm is only a slight extension from the left-upper-normal **trsm** work. Just as before, we know from the update

$$M_{12} := M_{11}^{-T} M_{12}$$

that M_{11} should be collected into a multiscalar, and M_{12} should be put in some form of row-wise multivector distribution. We can determine which particular multivector distribution by inspecting the following update,

$$M_{22} := M_{22} - M_{12}^{T} M_{12}.$$

While this update maps to a symmetric rank-k update (**syrk**) instead of a **gemm**, the only difference is that the local blocking strategy should take advantage of symmetry. Because it is still a panel-panel transposed-normal matrix-matrix multiply, we should gather $M_{12}^{\mathbb{N},\mathcal{C}}$ and $M_{12}^{\mathbb{N},\mathcal{R}}$ for the **syrk** update. Just as in the parallel left-upper-normal **trsm** algorithm, making M_{12} a row-major row-wise multivector for the **trsm** update allows us to prevent communicating in order to store the result in a matrix distribution.

Algorithm: $[M] := \text{CHOL\_BLK}(M)$

Partition $M \rightarrow \left(\begin{array}{c|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right)$

where M_{TL} is 0×0

while $m(M_{TL}) < m(M)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} M_{00} & M_{01} & M_{02} \\ \hline \star & M_{11} & M_{12} \\ \hline \star & \star & M_{22} \end{array}\right)$$

where M_{11} is $b \times b$

variant 1	**variant 2**	**variant 3**
$M_{01} := M_{00}^{-T} M_{01}$	$M_{11} := M_{11} - M_{01}^T M_{01}$	$M_{11} := \Gamma(M_{11})$
$\underline{M_{11} := M_{11} - M_{01}^T M_{01}}$	$M_{11} := \Gamma(M_{11})$	$M_{12} := M_{11}^{-T} M_{12}$
$M_{11} := \Gamma(M_{11})$	$M_{12} := M_{12} - M_{01}^T M_{02}$	$M_{22} := M_{22} - M_{12}^T M_{12}$
	$M_{12} := M_{11}^{-T} M_{12}$	

Continue with

$$\left(\begin{array}{c|c} M_{TL} & M_{TR} \\ \hline \star & M_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} M_{00} & M_{01} & M_{02} \\ \hline \star & M_{11} & M_{12} \\ \hline \star & \star & M_{22} \end{array}\right)$$

endwhile

Figure 5.3: Three blocked algorithms for performing an upper Cholesky factorization. Each is derived from a different Partitioned Matrix Expression (PME). The underlined update is not scalable.

The last step is to determine how to handle the update

$$M_{11} := \Gamma(M_{11}).$$

The operation is performed roughly n/b times, where b is the algorithmic blocksize, and costs $\frac{1}{3}b^3$ flops for each iteration. The overall cost is then roughly only $\frac{1}{3}nb^2$. Since we already need to collect M_{11} as a multiscalar for the **trsm** update, it is reasonable to redundantly compute the Cholesky factorization of M_{11} on all processors. The DAG for the chosen parallel updates is given in Fig. 5.4, and the implemented execution path is shown in Fig. 5.5.

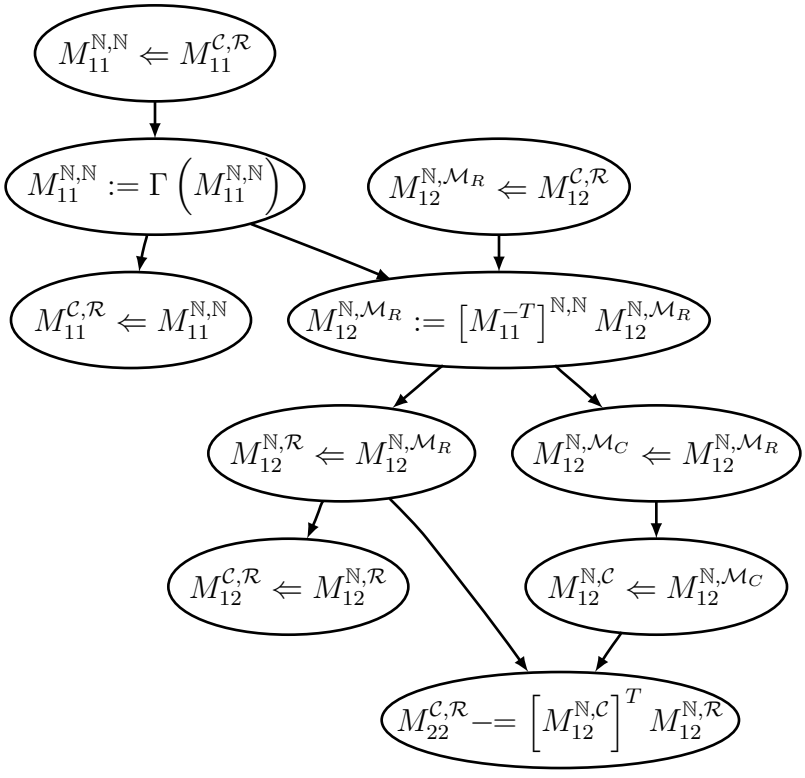

Figure 5.4: DAG for parallel Variant 3 Cholesky.

$M_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$M_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$M_{11}^{\mathbb{N},\mathbb{N}}$	$:=$	$\Gamma\left(M_{11}^{\mathbb{N},\mathbb{N}}\right)$	Local Cholesky
$\underline{M_{11}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$M_{11}^{\mathbb{N},\mathbb{N}}$	Filtered copy
$M_{12}^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$M_{12}^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$M_{12}^{\mathbb{N},\mathcal{M}_R}$	$:=$	$\left[M_{11}^{-T}\right]^{\mathbb{N},\mathbb{N}} M_{12}^{\mathbb{N},\mathcal{M}_R}$	`trsm`
$M_{12}^{\mathbb{N},\mathcal{R}}$	$\Leftarrow$	$M_{12}^{\mathbb{N},\mathcal{M}_R}$	Allgather over $\widehat{\mathcal{T}_C}$
$\underline{M_{12}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$M_{12}^{\mathbb{N},\mathcal{R}}$	Filtered copy
$M_{12}^{\mathbb{N},\mathcal{M}_C}$	$\Leftarrow$	$M_{12}^{\mathbb{N},\mathcal{M}_R}$	Send-recv in $\widehat{\mathcal{T}_W}$
$M_{12}^{\mathbb{N},\mathcal{C}}$	$\Leftarrow$	$M_{12}^{\mathbb{N},\mathcal{M}_C}$	Allgather over $\widehat{\mathcal{T}_R}$
$\underline{M_{22}^{\mathcal{C},\mathcal{R}}}$	$:=$	$M_{22}^{\mathcal{C},\mathcal{R}} - \left[M_{12}^{\mathbb{N},\mathcal{C}}\right]^T M_{12}^{\mathbb{N},\mathcal{R}}$	`gemm`s

Figure 5.5: Parallelization of an iteration of the Variant 3 Cholesky algorithm.

5.3 Parallelizing Variant 2

The parallelization of the Variant 2 algorithm is more complicated than the previous analysis because two of the updates are matrix-matrix multiplications. The first,

$$M_{11} := M_{11} - M_{01}^T M_{01},$$

is actually a symmetric rank-k update, but because $M_{11} \in \mathbb{R}^{b\times b}$, it is not necessarily beneficial to take advantage of symmetry. In fact, we avoid doing so in our parallel implementation. Whenever M_{01} is not empty, M_{01} is a column-panel and the first update should be parallelized as either a stationary A or stationary B matrix-matrix multiply. However, the second matrix-matrix

multiply update,

$$M_{12} := M_{12} - M_{01}^T M_{02},$$

should be parallelized using the previously discussed stationary B algorithm, as M_{02} is typically larger than M_{12} and M_{01}. We therefore decide to collect $M_{01}^{\mathcal{C},\mathbb{N}}$ and parallelize the update as

$$M_{12}^{\mathcal{C},\mathcal{R}} -= \sum_{q \in \mathcal{T}_C} \left[\left[M_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T M_{02}^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}}.$$

Since we have committed to gathering $M_{01}^{\mathcal{C},\mathbb{N}}$, we choose to reuse the distribution and parallelize the first update with the stationary B algorithm,

$$M_{12}^{\mathcal{C},\mathcal{R}} -= \sum_{q \in \mathcal{T}_C} \left[\left[M_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T M_{01}^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}}.$$

The remaining two updates are the Cholesky factorization of the diagonal block,

$$M_{11} := \Gamma(M_{11}),$$

and the **trsm**,

$$M_{12} := M_{11}^{-1} M_{12}.$$

As before, we parallelize the **trsm** by putting M_{12} into a row-wise multivector distribution and collecting M_{11} as a multiscalar. Since there is no opportunity to reuse the multivector distribution, the choice of a particular multivector distribution is arbitrary. We again choose to redundantly compute the Cholesky factorization of M_{11} on all nodes, and the resulting DAG and algorithm are shown in Figs. 5.6 and 5.7.

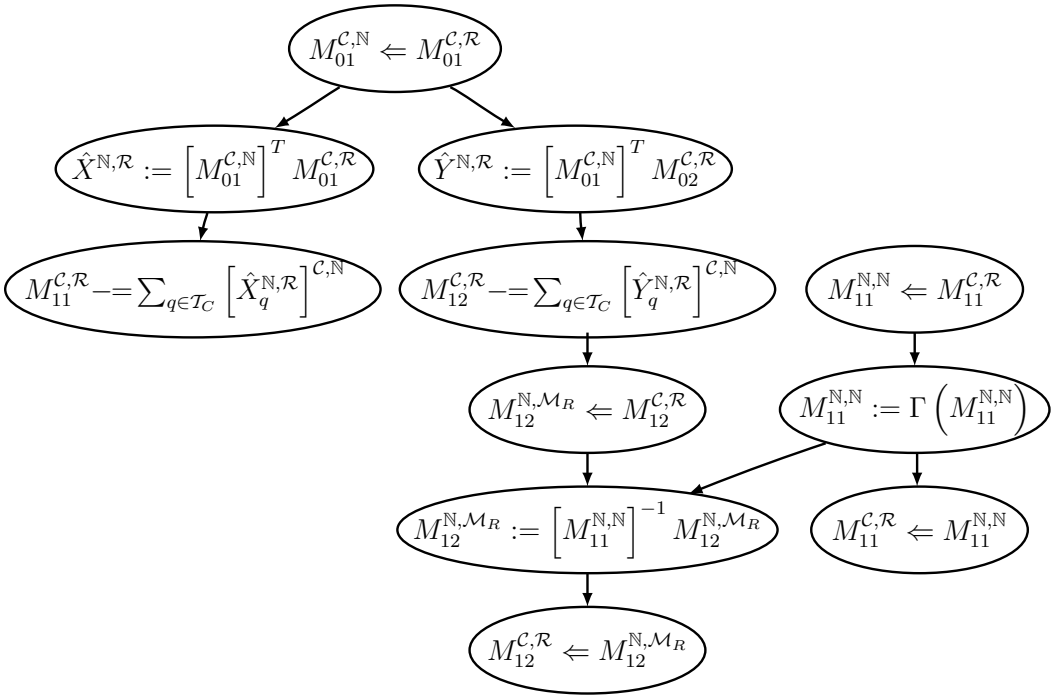

Figure 5.6: DAG for parallel Variant 2 Cholesky.

5.4 Performance Results

The Variant 2 and 3 parallel algorithms were implemented using C++ and MPI, and then tested on 16 and 64 cores of Lonestar. Performance for both variants is benchmarked against ScaLAPACK in Fig. 5.8, where ScaLAPACK is shown to outperform the Variant 3 implementation as the matrix size increases past 25,000. However, the Variant 2 implementation uniformly outperforms both ScaLAPACK and Variant 3, although Variant 2 is by far more difficult to implement. The performance gain is due to Variant 2 casting its computation in terms of **gemm**s instead of a **syrk**, which is harder to map directly to serial BLAS calls in an elemental scheme. At a problem size

$M_{01}^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$M_{01}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\hat{X}^{\mathbb{N},\mathcal{R}}$	$:=$	$\left[M_{01}^{\mathcal{C},\mathbb{N}}\right]^T M_{01}^{\mathcal{C},\mathcal{R}}$	`gemm`
$M_{11}^{\mathcal{C},\mathcal{R}}$	$-=$	$\sum_{q\in\mathcal{T}_C}\left[\hat{X}_q^{\mathbb{N},\mathcal{R}}\right]^{\mathcal{C},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$
$M_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$M_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$M_{11}^{\mathbb{N},\mathbb{N}}$	$:=$	$\Gamma\left(M_{11}^{\mathbb{N},\mathbb{N}}\right)$	Local Cholesky
$\underline{M_{11}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$M_{11}^{\mathbb{N},\mathbb{N}}$	Filtered copy
$\hat{Y}^{\mathbb{N},\mathcal{R}}$	$:=$	$\left[M_{01}^{\mathcal{C},\mathbb{N}}\right]^T M_{02}^{\mathcal{C},\mathcal{R}}$	`gemm`
$M_{12}^{\mathcal{C},\mathcal{R}}$	$-=$	$\sum_{q\in\mathcal{T}_C}\left[\hat{Y}_q^{\mathbb{N},\mathcal{R}}\right]^{\mathcal{C},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$
$M_{12}^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$M_{12}^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$M_{12}^{\mathbb{N},\mathcal{M}_R}$	$:=$	$\left[M_{11}^{\mathbb{N},\mathbb{N}}\right]^{-1} M_{12}^{\mathbb{N},\mathcal{M}_R}$	`trsm`
$\underline{M_{12}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$M_{12}^{\mathbb{N},\mathcal{M}_R}$	Allgather over $\widehat{\mathcal{T}_C}$

Figure 5.7: Parallelization of an iteration of the Variant 2 Cholesky algorithm.

of 15,000, Variant 2 performance is shown to be more than 25% higher than ScaLAPACK on both 16 and 64 cores.

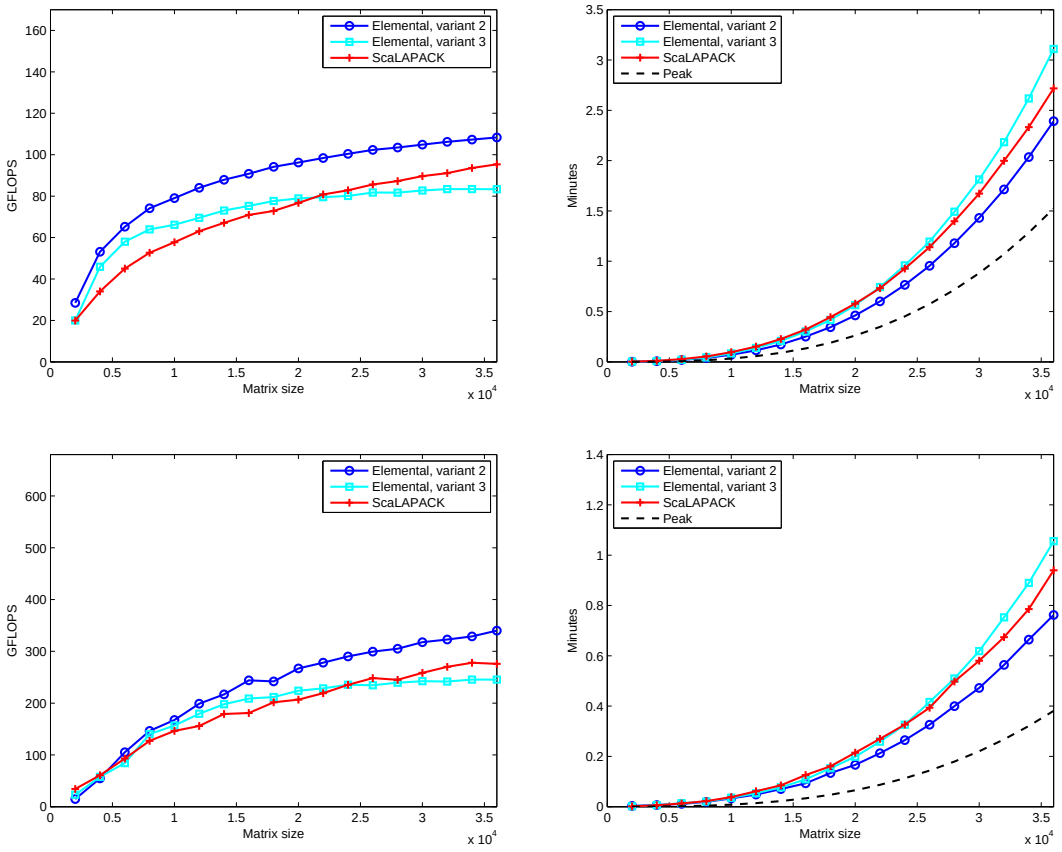

Figure 5.8: Performance and wall-clock time of parallel Cholesky factorizations on 16 (top) and 64 (bottom) cores of Lonestar.

Chapter 6

Reduction to Symmetric Standard EVP

The last tool needed for our generalized eigensolution is a scalable parallel algorithm for reducing the generalized EVP to standard form. This reduction is accomplished through forming a matrix A, which is defined in (1.7) as

$$A = U^{-T} K U^{-1},$$

where $U = \Gamma(M)$, and K is symmetric. A naïve approach would be to ignore the symmetry of the result and simply perform two successive parallel **trsm**s, which has a flop count of $2n^3 + \mathcal{O}(n^2)$. While the standard algorithm for this reduction, **sygst**, has a flop count of $n^3 + \mathcal{O}(n^2)$, we will show that one of the steps of the algorithm is inherently unscalable and present an alternative algorithm that avoids the scalability issue. All three approaches are then benchmarked with some surprising results.

6.1 Original and Revised Algorithms

We begin by rearranging (1.7) into the form

$$K = U^T A U,$$

and partitioning the equation into 2×2 matrices of blocks. Then,

$$\left(\begin{array}{c|c} K_{TL} & K_{TR} \\ \hline \star & K_{BR} \end{array}\right) = \left(\begin{array}{c|c} U_{TL}^T & 0 \\ \hline U_{TR}^T & U_{BR}^T \end{array}\right) \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & A_{BR} \end{array}\right) \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$$
$$= \left(\begin{array}{c|c} U_{TL}^T A_{TL} U_{TL} & U_{TL}^T \left(A_{TL} U_{TR} + A_{TR} U_{BR}\right) \\ \hline \star & U_{BR}^T A_{BR} U_{BR} + U_{TR}^T A_{TL} U_{TR} + U_{BR}^T A_{TR}^T U_{TR} + U_{TR}^T A_{TR} U_{BR} \end{array}\right),$$

where we have again used '$\star$' to denote an unaccessed portion of a symmetric matrix. The algorithm used by LAPACK for the reduction, shown in the left column of Fig. 6.1, has the loop-invariant

$$\left(\begin{array}{c|c} K_{TL} & K_{TR} \\ \hline \star & K_{BR} \end{array}\right) = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline \star & U_{BR}^T A_{BR} U_{BR} \end{array}\right),$$

and we will use $\hat{K}$ to represent the original state of the input matrix, K. From the step

$$K_{12} := K_{12} U_{22}^{-1},$$

it is clear that a triangular solve with the number of "right-hand sides", the rows of K_{12}, equal to the algorithmic blocksize must be performed. Since this is an inherently unscalable operation and constitutes roughly 1/3 of the flops of the routine, it is expected that this algorithmic variant will perform poorly. If we instead choose the loop-invariant

$$\left(\begin{array}{c|c} K_{TL} & K_{TR} \\ \hline \star & K_{BR} \end{array}\right) = \left(\begin{array}{c|c} A_{TL} & U_{TL}^{-T} \hat{K}_{TR} \\ \hline \star & \hat{K}_{BR} \end{array}\right),$$

then we arrive at the algorithm shown in the right column of Fig. 6.1, which has eliminated the unscalable **trsm** update.

Algorithm: $K := U^{-T} K U^{-1}$

Partition $K \rightarrow \left(\begin{array}{c|c} K_{TL} & K_{TR} \\ \hline \star & K_{BR} \end{array}\right)$, U similarly

where K_{TL} is 0×0, U_{TL} is 0×0

while $m(K_{TL}) < m(K)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} K_{TL} & K_{TR} \\ \hline \star & K_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} K_{00} & K_{01} & K_{02} \\ \hline \star & K_{11} & K_{12} \\ \hline \star & \star & K_{22} \end{array}\right), \text{ U similarly}$$

where K_{11} is $b \times b$, U_{11} is $b \times b$

standard	**revised**
$K_{11} := U_{11}^{-T} K_{11} U_{11}^{-1}$	$X := K_{00} U_{01}$
$K_{12} := U_{11}^{-T} K_{12}$	$K_{01} := K_{01} - X$
$K_{12} := K_{12} - \frac{1}{2} K_{11} U_{12}$	$K_{11} := K_{11} - K_{01}^T U_{01} - U_{01}^T K_{01}$
$K_{22} := K_{22} - K_{12}^T U_{12} - U_{12}^T K_{12}$	$K_{11} := K_{11} - U_{01}^T X$
$K_{12} := K_{12} - \frac{1}{2} K_{11} U_{12}$	$K_{11} := U_{11}^{-T} K_{11} U_{11}^{-1}$
$\underline{K_{12} := K_{12} U_{22}^{-1}}$	$K_{01} := K_{01} U_{11}^{-1}$
	$K_{12} := K_{12} - U_{01}^T K_{02}$
	$K_{12} := U_{11}^{-T} K_{12}$

Continue with

$$\left(\begin{array}{c|c} K_{TL} & K_{TR} \\ \hline \star & K_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} K_{00} & K_{01} & K_{02} \\ \hline \star & K_{11} & K_{12} \\ \hline \star & \star & K_{22} \end{array}\right), \text{ U similarly}$$

endwhile

Figure 6.1: The standard and revised **sygst** algorithms. The poorly parallelizable update in the standard algorithm has been underlined.

6.2 Parallelizing the Revised Algorithm

The first update to investigate is

$$X := K_{00}U_{01},$$

where K_{00} is symmetric and stored in the upper triangle. The update is then more accurately written as

$$X := \text{triu}(K_{00})U_{01} + \text{trisu}(K_{00})^T U_{01},$$

where triu$(\cdot)$ and trisu$(\cdot)$ respectively return their input matrices with only the upper triangular and strictly upper triangular portions left nonzero. The update should then be parallelized as the sum of normal-normal and transposed-normal stationary A matrix-matrix multiplications, such that

$$X^{\mathcal{C},\mathcal{R}} := \sum_{q\in\mathcal{T}_R} \left[\text{triu}(K_{00})^{\mathcal{C},\mathcal{R}^q} U_{01}^{\mathcal{R}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}} + \left[\sum_{q\in\mathcal{T}_C} \left[\left[\text{trisu}(K_{00})^{\mathcal{C}^q,\mathcal{R}}\right]^T U_{01}^{\mathcal{C}^q,\mathbb{N}}\right]^{\mathbb{N},\mathcal{C}}\right]^{\leftrightarrow}.$$

In order to do so, we are required to form both $U_{01}^{\mathcal{C},\mathbb{N}}$ and $U_{01}^{\mathcal{R},\mathbb{N}}$.

The second update is trivially parallelized as

$$K_{01}^{\mathcal{C},\mathcal{R}} := K_{01}^{\mathcal{C},\mathcal{R}} - X^{\mathcal{C},\mathcal{R}},$$

so we move on to the third update,

$$K_{11} := K_{11} - K_{01}^T U_{01} - U_{01}^T K_{01},$$

which is a symmetric rank-$2k$ update (**syr2k**). However, the local update is in general not even square, and we avoid taking advantage of the global symmetry because K_{11} is $n_b \times n_b$. We then decide to parallelize both $K_{01}^T U_{01}$ and

$U_{01}^T K_{01}$ using transposed-normal stationary B algorithms because it requires less communication than the stationary A approach. Additionally, we have the added benefit that we are already required to collect $U_{01}^{\mathcal{C},\mathbb{N}}$ and can simply reuse it. The parallel update can be expressed as

$$\begin{aligned} K_{11}^{\mathcal{C},\mathcal{R}} &:= \sum_{q \in \mathcal{T}_C} \left[\left[K_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T U_{01}^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}} + \sum_{q \in \mathcal{T}_C} \left[\left[U_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T K_{01}^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}} \\ &= \sum_{q \in \mathcal{T}_C} \left[\left[K_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T U_{01}^{\mathcal{C}^q,\mathcal{R}} + \left[U_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T K_{01}^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}}, \end{aligned}$$

and we are now required to also collect $K_{01}^{\mathcal{C},\mathbb{N}}$. In fact, we can use the same technique to combine the parallelization of this update with the next one,

$$K_{11} := K_{11} - U_{01}^T X,$$

so that we form the update as

$$K_{11}^{\mathcal{C},\mathcal{R}} := K_{11}^{\mathcal{C},\mathcal{R}} - \sum_{q \in \mathcal{T}_C} \left[\left[K_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T U_{01}^{\mathcal{C}^q,\mathcal{R}} + \left[U_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T K_{01}^{\mathcal{C}^q,\mathcal{R}} + \left[U_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T X^{\mathcal{C}^q,\mathcal{R}} \right]^{\mathcal{C},\mathbb{N}}.$$

An astute reader will immediately notice that we can save a few flops by grouping the update as

$$K_{11}^{\mathcal{C},\mathcal{R}} := K_{11}^{\mathcal{C},\mathcal{R}} - \sum_{q \in \mathcal{T}_C} \left[\left[K_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T U_{01}^{\mathcal{C}^q,\mathcal{R}} + \left[U_{01}^{\mathcal{C}^q,\mathbb{N}} \right]^T \left(K_{01}^{\mathcal{C}^q,\mathcal{R}} + X^{\mathcal{C}^q,\mathcal{R}} \right) \right]^{\mathcal{C},\mathbb{N}}.$$

In fact, recognizing that the previous update was to subtract X from K_{01}, we can avoid the operation $K_{01}^{\mathcal{C},\mathcal{R}} + X^{\mathcal{C},\mathcal{R}}$ by computing $\left[U_{01}^{\mathcal{C},\mathbb{N}} \right]^T K_{01}^{\mathcal{C},\mathcal{R}}$ before-hand.

The next update,

$$K_{11} := U_{11}^T K_{11} U_{11}^{-1},$$

should be redundantly computed in the same manner as the factorizations of diagonal blocks in our parallel Cholesky algorithm. We then collect $K_{11}^{\mathbb{N},\mathbb{N}}$ and $U_{11}^{\mathbb{N},\mathbb{N}}$ so that we may redundantly compute

$$K_{11}^{\mathbb{N},\mathbb{N}} := \left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-T} K_{11}^{\mathbb{N},\mathbb{N}} \left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-1}.$$

The following operation is

$$K_{01} := K_{01} U_{11}^{-1},$$

which is performed using the same technique as all previous **trsm**s. Because we do not need to redistribute K_{01} for any other updates, the choice of which column-wise multivector distribution to use for K_{01} is arbitrary. The next-to-last operation,

$$K_{12} := K_{12} - U_{01}^{T} K_{02},$$

should be performed using a stationary B parallel matrix-matrix multiplication algorithm. We have already required the collection of $U_{01}^{\mathbb{N},\mathcal{C}}$, so the only communication necessary is in reducing the result across the team.

Finally, we have the **trsm**,

$$K_{12} := U_{11}^{-T} K_{12}.$$

We have already gathered $U_{11}^{\mathbb{N},\mathbb{N}}$, and this is the only update that requires redistributing K_{12}, so the required redistribution is to put K_{12} into an arbitrary row-wise multivector. The resulting DAG for the parallel updates in shown in Fig. 6.2, and a legal execution path is shown in Fig. 6.3.

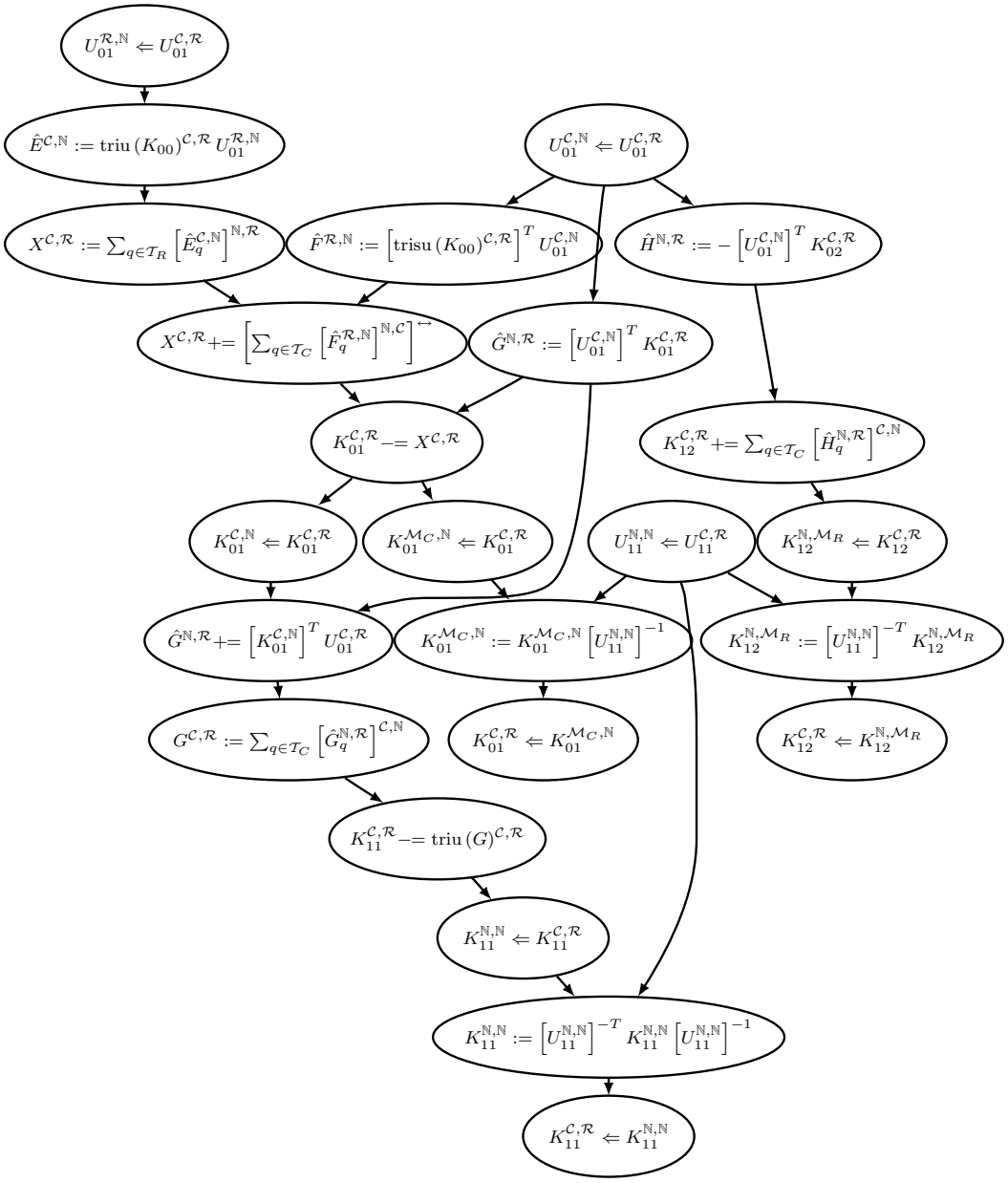

Figure 6.2: DAG for parallel revised **sygst** algorithm.

$U_{01}^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$U_{01}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$U_{01}^{\mathcal{R},\mathbb{N}}$	$\Leftarrow$	$U_{01}^{\mathcal{C},\mathcal{R}}$	All-to-all $\widehat{\mathcal{T}_R}$, Send-recv, Allgather $\widehat{\mathcal{T}_C}$
$\hat{G}^{\mathbb{N},\mathcal{R}}$	$:=$	$\hat{G}^{\mathbb{N},\mathcal{R}} + \left[U_{01}^{\mathcal{C},\mathbb{N}}\right]^T K_{01}^{\mathcal{C},\mathcal{R}}$	`gemm`
$\hat{E}^{\mathcal{C},\mathbb{N}}$	$:=$	$\mathrm{triu}\left(K_{00}\right)^{\mathcal{C},\mathcal{R}} U_{01}^{\mathcal{R},\mathbb{N}}$	`gemms`
$X^{\mathcal{C},\mathcal{R}}$	$:=$	$\sum_{q\in\mathcal{T}_R}\left[\hat{E}_q^{\mathcal{C},\mathbb{N}}\right]^{\mathbb{N},\mathcal{R}}$	Reduce-scatter over $\widehat{\mathcal{T}_R}$
$\hat{F}^{\mathcal{R},\mathbb{N}}$	$:=$	$\left[\mathrm{trisu}\left(K_{00}\right)^{\mathcal{C},\mathcal{R}}\right]^T U_{01}^{\mathcal{C},\mathbb{N}}$	`gemms`
$X^{\mathcal{C},\mathcal{R}}$	$:=$	$X^{\mathcal{C},\mathcal{R}} + \left[\sum_{q\in\mathcal{T}_C}\left[\hat{F}_q^{\mathcal{R},\mathbb{N}}\right]^{\mathbb{N},\mathcal{C}}\right]^{\leftrightarrow}$	Reduce-scatter $\widehat{\mathcal{T}_C}$, transpose dist.
$K_{01}^{\mathcal{C},\mathcal{R}}$	$:=$	$K_{01}^{\mathcal{C},\mathcal{R}} - X^{\mathcal{C},\mathcal{R}}$	`axpy`s
$K_{01}^{\mathcal{C},\mathbb{N}}$	$\Leftarrow$	$K_{01}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_R}$
$\hat{G}^{\mathbb{N},\mathcal{R}}$	$+=$	$\left[K_{01}^{\mathcal{C},\mathbb{N}}\right]^T U_{01}^{\mathcal{C},\mathcal{R}}$	`gemm`
$G^{\mathcal{C},\mathcal{R}}$	$:=$	$\sum_{q\in\mathcal{T}_C}\left[\hat{G}_q^{\mathbb{N},\mathcal{R}}\right]^{\mathcal{C},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$
$K_{11}^{\mathcal{C},\mathcal{R}}$	$:=$	$K_{11}^{\mathcal{C},\mathcal{R}} - \mathrm{triu}\left(G\right)^{\mathcal{C},\mathcal{R}}$	`axpy`s
$U_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$U_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$K_{01}^{\mathcal{M}_C,\mathbb{N}}$	$\Leftarrow$	$K_{01}^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$K_{01}^{\mathcal{M}_C,\mathbb{N}}$	$:=$	$K_{01}^{\mathcal{M}_C,\mathbb{N}}\left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-1}$	`trsm`
$\underline{K_{01}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$K_{01}^{\mathcal{M}_C,\mathbb{N}}$	All-to-all over $\widehat{\mathcal{T}_R}$
$K_{11}^{\mathbb{N},\mathbb{N}}$	$\Leftarrow$	$K_{11}^{\mathcal{C},\mathcal{R}}$	Allgather over $\widehat{\mathcal{T}_W}$
$K_{11}^{\mathbb{N},\mathbb{N}}$	$:=$	$\left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-T} K_{11}^{\mathbb{N},\mathbb{N}}\left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-1}$	`sygst`
$\underline{K_{11}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$K_{11}^{\mathbb{N},\mathbb{N}}$	Filtered copy
$\hat{H}^{\mathbb{N},\mathcal{R}}$	$:=$	$-\left[U_{01}^{\mathcal{C},\mathbb{N}}\right]^T K_{02}^{\mathcal{C},\mathcal{R}}$	`gemm`
$K_{12}^{\mathcal{C},\mathcal{R}}$	$:=$	$K_{12}^{\mathcal{C},\mathcal{R}} + \sum_{q\in\mathcal{T}_C}\left[\hat{H}_q^{\mathbb{N},\mathcal{R}}\right]^{\mathcal{C},\mathbb{N}}$	Reduce-scatter over $\widehat{\mathcal{T}_C}$
$K_{12}^{\mathbb{N},\mathcal{M}_R}$	$\Leftarrow$	$K_{12}^{\mathcal{C},\mathcal{R}}$	All-to-all over $\widehat{\mathcal{T}_C}$
$K_{12}^{\mathbb{N},\mathcal{M}_R}$	$:=$	$\left[U_{11}^{\mathbb{N},\mathbb{N}}\right]^{-T} K_{12}^{\mathbb{N},\mathcal{M}_R}$	`trsm`
$\underline{K_{12}^{\mathcal{C},\mathcal{R}}}$	$\Leftarrow$	$K_{12}^{\mathbb{N},\mathcal{M}_R}$	All-to-all over $\widehat{\mathcal{T}_C}$

Figure 6.3: Parallelization of a single iteration of the revised **sygst** algorithm.

6.3 Performance Results

The parallelization of the revised **sygst** algorithm was implemented in the same manner as before, and tested on 16 and 64 cores of Lonestar for problem sizes ranging between 2000 and $34,000$. Instead of simply benchmarking the elemental revised reduction routine against the ScaLAPACK reduction routine, comparisons were also made using the naïve approach to the reduction, two parallel **trsm**s. Because the naïve approach incurs twice as many flops as the standard and revised algorithms, its performance numbers were halved to normalize against the symmetric reduction methods. It is important to again note that using two **trsm**s avoids taking advantage of symmetry and results in the loss of any data stored in the lower half of the input matrix.

Results are shown in Fig. 6.4, and the simple approach is shown to do surprisingly well. Though the elemental revised algorithm outperforms the other three implementations for all cases, the simple elemental approach only slightly lags on 64 cores. In fact, on both 16 and 64 cores, the elemental naïve performance dominates that of both ScaLAPACK implementations. Additionally, at a problem size of $15,000$ on 64 cores, both elemental approaches nearly double the performance of the ScaLAPACK **sygst**.

Using the ScaLAPACK **trsm**s, the simple approach is shown to be inferior to the ScaLAPACK **sygst** for large problem sizes on 16 cores, but it is uniformly superior on 64 cores, where it rapidly approaches its asymptotic performance level. The reason for the relative degradation in performance of the ScaLAPACK **sygst** is almost certainly the use of the unscalable triangular

solve with the number of right-hand sides set to the algorithmic blocksize.

For large numbers of processes, it is likely that the wall-clock time of the naïve elemental algorithm will be indistinguishable from the significantly more complicated algorithm. We then conclude that the new parallel reduction algorithm improves performance for arbitrary numbers of cores if auxilliary data is stored in the lower-half of the matrix, but the benefit is negligible for large numbers of cores when the entire matrix can be overwritten.

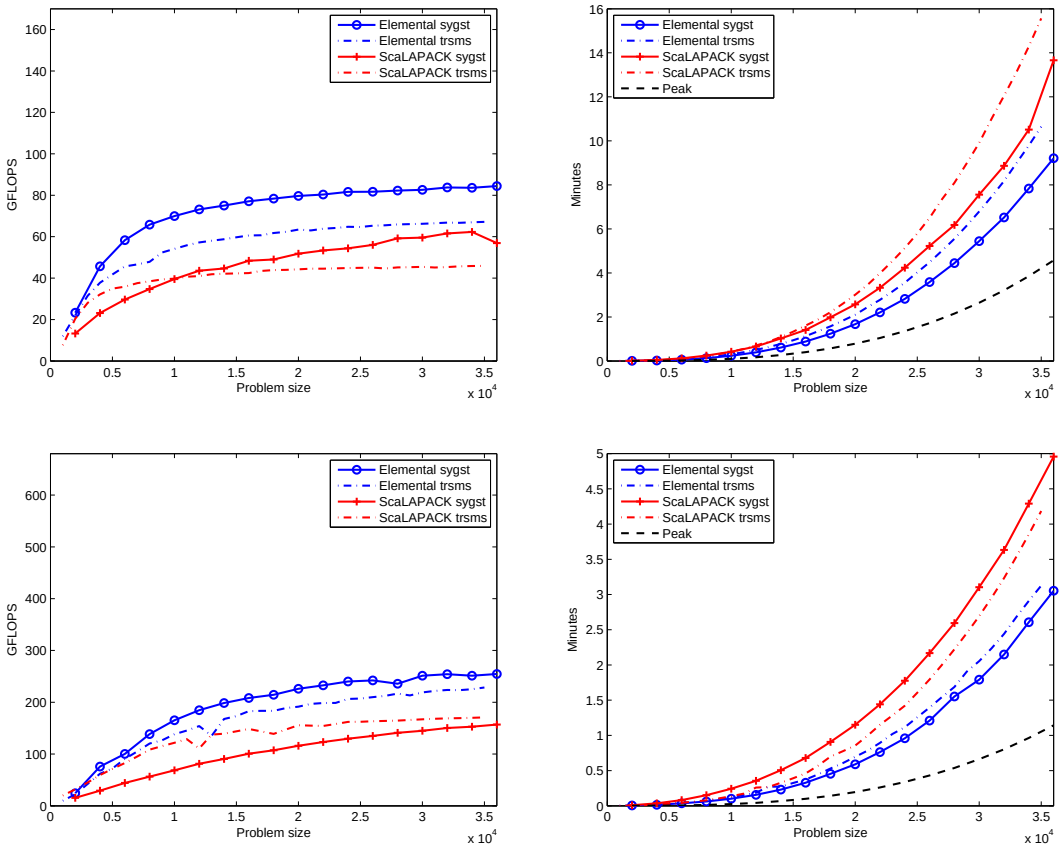

Figure 6.4: Performance and wall-clock time of parallel **sygst** on 16 (top) and 64 (bottom) cores of Lonestar

Chapter 7

Conclusions

The use of the presented theory for deriving parallel algorithms has been shown to significantly increase the performance of all of the wrapper routines for extending a parallel standard eigensolution to the generalized EVP. A comparison of the $\mathcal{O}(n^3)$ computational costs of the overall eigensolution is shown in Fig. 7.1, and we note that the tridiagonal eigensolution is $\mathcal{O}(n^2)$. Though the discussed wrapper routines comprise less than half of the overall computation for the generalized eigensolution, the techniques developed for parallel matrix-matrix multiplication were not extended to the application of WY transforms. Because applying WY transforms is the most computationally expensive step, improving its performance is clearly very beneficial and is left as future work.

The combined wall-clock times and performance numbers from Cholesky factorization, the reduction to standard form, and the left-upper-normal **trsm** on 16 and 64 cores are shown in Figs. 7.2 and 7.3 versus the equivalent ScaLAPACK operations. While there was an increase in performance for Cholesky factorizations, the main improvements are in the algorithms for the reduction to the standard eigenvalue problem and the back-transformation of the eigen-

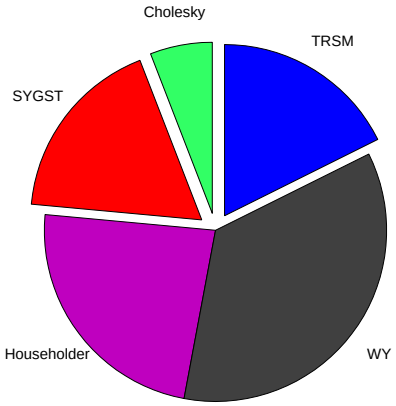

Figure 7.1: The distribution of flops in the proposed generalized eigensolution method. The routines that extend the standard eigensolution to generalized form have been emphasized.

vectors to the generalized problem. Not only were there larger improvements in the percent of peak attained by these two algorithms, but their flop-counts are three times larger than for a Cholesky factorization.

Overall performance on both 16 and 64 cores shows the elemental approach having a large advantage over the entire range of problem sizes. At a problem size of $34,000$, switching from ScaLAPACK to the elemental implementations roughly drops the wall-clock time from 20 minutes to 15 minutes on 16 cores, and from 7 minutes to 5 minutes on 64 cores. On 64 cores, the improved reduction algorithm was shown to be only slightly faster than using successive left-upper-transposed and right-upper-normal parallel **trsm**. Additionally, the back-transformation implementation achieved large performance gains, and is simply a parallel left-upper-normal **trsm**. The majority of our performance can then be reached through the presented algorithms for parallel

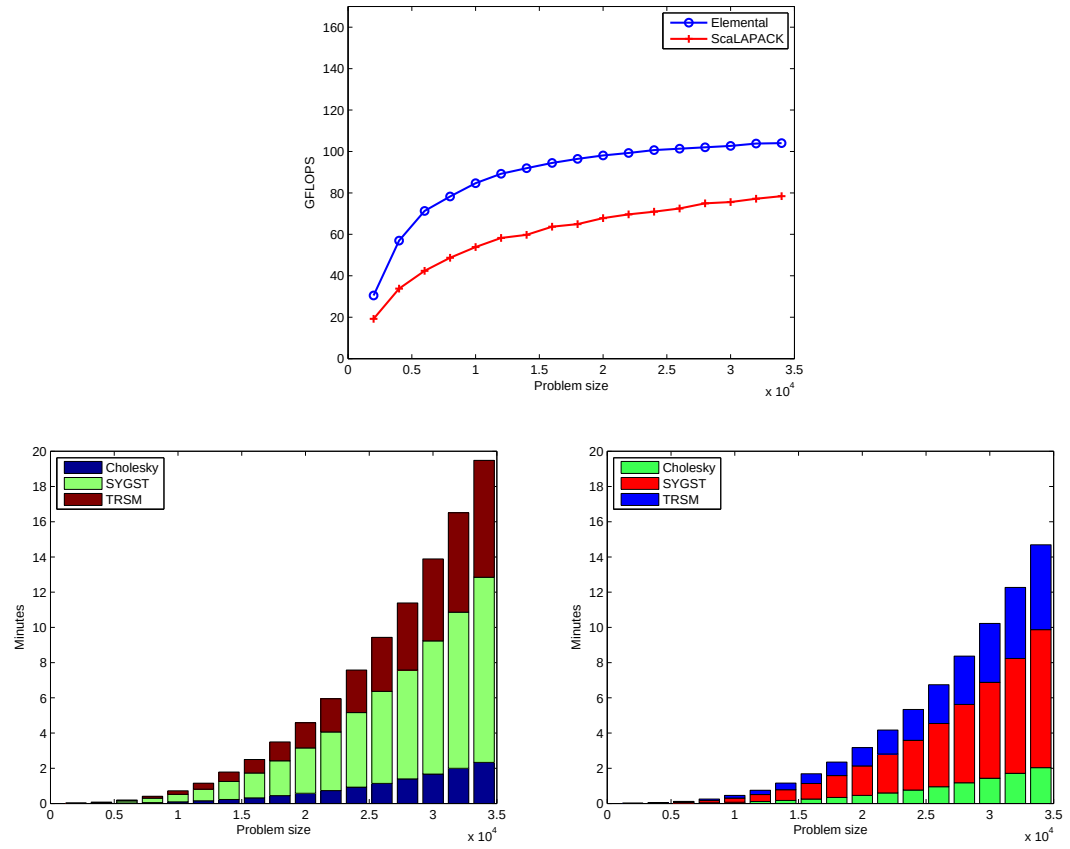

Figure 7.2: The total wall-clock time, on 16 cores of Lonestar, of the added routines for the generalized EVP using ScaLAPACK (left) and the elemental approach (right).

trsm, though this approach destroys the ability to store information in the lower triangle of K.

The use of the presented theory clearly reaches well outside of the realm of eigensolutions, as the general goal when deriving a parallel linear algebra algorithm is to cast as much computation as possible into parallel matrix-matrix multiplication. Additionally, performance results strongly suggest that

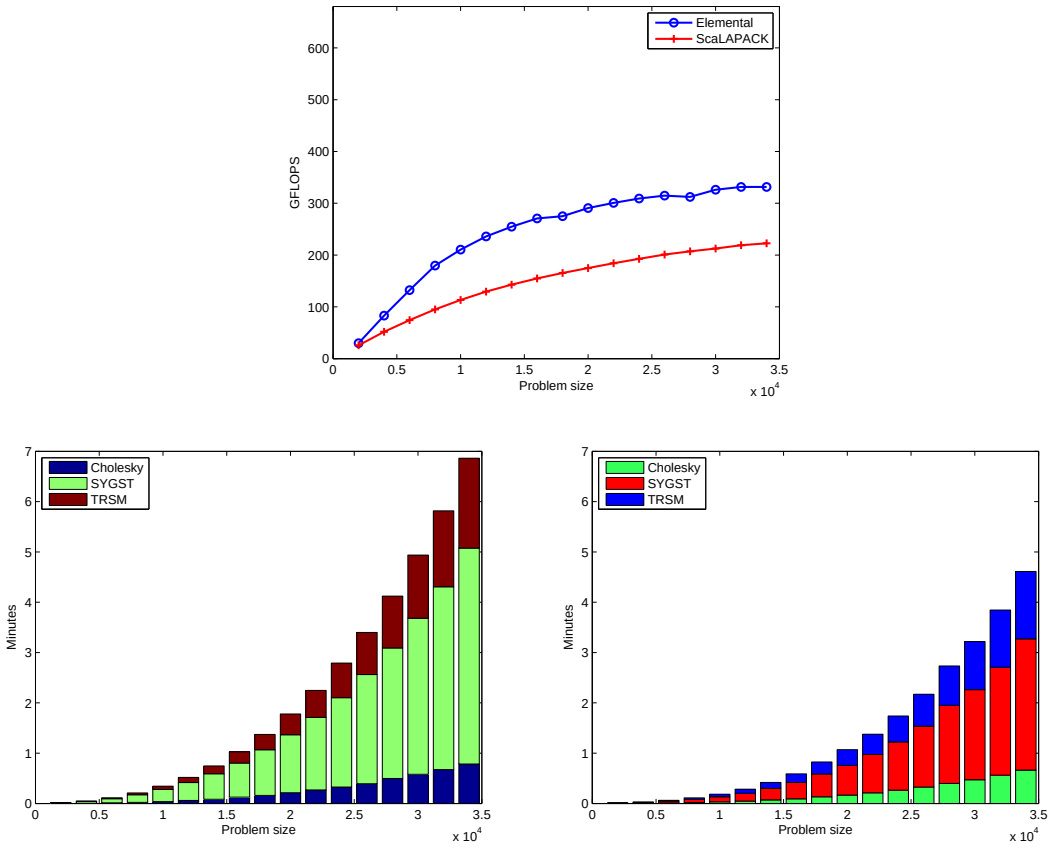

Figure 7.3: The total wall-clock time, on 64 cores of Lonestar, of the added routines for the generalized EVP using ScaLAPACK (left) and the elemental approach (right).

elemental distributions allow for faster implementations of the Level 3 BLAS than those resulting from blocked distributions.

Bibliography

[1] Philip Alpatov, Greg Baker, Carter Edwards, John Gunnels, Greg Morrow, James Overfelt, Robert van de Geijn, and Yuan-Jye J. Wu. PLAPACK: Parallel Linear Algebra Package design overview. In *Supercomputing '97: Proceedings of the 1997 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 1–16, New York, NY, USA, 1997. ACM.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK's User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[3] Paolo Bientinesi, Inderjit S. Dhillon, and Robert A. van de Geijn. A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations. *SIAM J. Sci. Comput.*, 27(1):43–66, 2005.

[4] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 31(1):1–26, March 2005.

[5] 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*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[6] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: Theory, practice, and experience. *Concurr. Comput. : Pract. Exper.*, 19(13):1749–1783, 2007.

[7] Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, and Robert van de Geijn. Parallel implementation of BLAS: General techniques for level 3 BLAS. Technical report, Austin, TX, USA, 1995.

[8] Inderjit S. Dhillon. *A New O (N(2)) Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, Berkeley, CA, USA, 1998.

[9] Inderjit S. Dhillon and Beresford N. Parlett. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and Appl*, 387:1–28, 2004.

[10] Inderjit S. Dhillon, Beresford N. Parlett, and Christof Vömel. The design and implementation of the MRRR algorithm. *ACM Trans. Math. Softw.*, 32(4):533–560, 2006.

[11] Carter Edwards, Po Geng, Abani Patra, and Robert van de Geijn. Parallel matrix distributions: Have we been doing it all wrong? Technical report, Austin, TX, USA, 1995.

[12] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, October 1996.

[13] Kazushige Goto and Robert A. van de Geijn. Anatomy of a high-performance matrix multiplication. *ACM Transactions on Mathematical Software*, 34(3):12, May 2008. Article 12, 25 pages.

[14] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI (2nd ed.): Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, USA, 1999.

[15] J. Gunnels, C. Lin, G. Morrow, and R. van de Geijn. A flexible class of parallel matrix multiplication algorithms. *Parallel Processing Symposium, International*, 0:0110, 1998.

[16] Bruce Hendrickson, Elizabeth Jessup, and Christopher Smith. Toward an efficient parallel eigensolver for dense symmetric matrices. *SIAM J. Sci. Comput.*, 20(3):1132–1154, 1999.

[17] Bruce A. Hendrickson and David E. Womble. The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM J. Sci. Comput.*, 15(5):1201–1226, 1994.

[18] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[19] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Ortí, Robert van de Geijn, and Field G. Van Zee. Accumulating householder transformations, revisited. *ACM Trans. Math. Softw.*, 32(2):169–179, 2006.

[20] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia,PA, 1998.

[21] Jelena Pješivac-Grbović, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, and Jack J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing*, 10(2):127–143, 2007.

[22] G. W. Stewart. Communication and matrix computations on large message passing systems. *Parallel Computing*, 16(1):27–40, 1990.

[23] Sayantan Sur, Matthew J. Koop, and Dhabaleswar K. Panda. High-performance and scalable MPI over InfiniBand with reduced memory usage: an in-depth performance analysis. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 105, New York, NY, USA, 2006. ACM.

[24] Robert A. van de Geijn, Philip Alpatou, Greg Baker, Carter Edwards, John Gunnels, Greg Morrow, and James Overfelt. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA, USA, 1997.

[25] Robert A. van de Geijn and Enrique S. Quintana-Ortí. *The Science of Programming Matrix Computations*. `www.lulu.com`, 2008.

Vita

Jack Lesly Poulson was born in Bryan, Texas on 17 March 1986, the son of Gary E. Poulson and Jacqueline C. Jones. He received his Bachelor of Science in Aerospace Engineering from The University of Texas at Austin in May 2007 and entered their masters program in Aerospace Engineering in the Fall of 2007. He will be working on high-performance algorithms at Microsoft Research in Redmond, WA over the summer of 2009, and entering UT's Computational Science, Engineering, and Mathematics doctoral program as a fellow in the Fall of 2009.

Permanent address: 15615 Boulder Oaks Dr.
Houston, Texas 77084

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.