A Communication-Avoiding Parallel Algorithm for the Symmetric Eigenvalue Problem

Edgar Solomonik
University of Illinois at Urbana-Champaign
Department of Computer Science

Grey Ballard
Wake Forest University
Department of Computer Science

James Demmel
University of California, Berkeley
Department of Mathematics & Department of Electrical Engineering and Computer Science

Torsten Hoefler
ETH Zurich
Computer Science Department

ABSTRACT

Many large-scale scientific computations require eigenvalue solvers in a scaling regime where efficiency is limited by data movement. We introduce a parallel algorithm for computing the eigenvalues of a dense symmetric matrix, which performs asymptotically less communication than previously known approaches. We provide analysis in the Bulk Synchronous Parallel (BSP) model with additional consideration for communication between a local memory and cache. Given sufficient memory to store c copies of the symmetric matrix, our algorithm requires $\Theta(\sqrt{c})$ less interprocessor communication than previously known algorithms, for any $c \leq p^{1/3}$ when using p processors. The algorithm first reduces the dense symmetric matrix to a banded matrix with the same eigenvalues. Subsequently, the algorithm employs successive reduction to $O(\log p)$ thinner banded matrices. We employ two new parallel algorithms that achieve lower communication costs for the full-to-band and band-to-band reductions. Both of these algorithms leverage a novel QR factorization algorithm for rectangular matrices.

1 INTRODUCTION

The eigenvalue decomposition of a symmetric matrix A is $A = UDU^T$ where D is a diagonal matrix of eigenvalues and the columns of the orthogonal matrix U are the eigenvectors of A. Dense symmetric eigensolvers typically reduce the matrix to a tridiagonal matrix with the same eigenvalues, compute the eigenvalues D of this tridiagonal matrix [17], and, if desired, apply the orthogonal transformation backwards to compute the eigenvectors U. Although algorithms for tridiagonalizing a symmetric matrix require the same asymptotic amount of work as one-sided decompositions such as LU and QR factorization, they have a more complex dependency structure, which makes communication-efficient parallelization challenging. Efficient execution of scientific applications such as electronic structure methods, which compute eigenvalue decompositions of a sequence of symmetric matrices (see, e.g. Hartree-Fock method [21, 26]), requires scalable symmetric eigensolvers.

We analyze the scalability of parallel algorithms in a Bulk Synchronous Parallel (BSP) cost model [39]. In addition to quantifying horizontal communication (data movement between processors) and synchronization, we augment the BSP model with an additional bandwidth cost parameter for vertical communication (data movement between memory and cache). There are known algorithms for Cholesky, LU, and QR factorization [3, 35, 38], which for $n \times n$ input matrices on a p-processor system, have horizontal communication complexity $W = O(n^2/\sqrt{cp})$, require $S = O(\sqrt{cp})$ synchronizations, and use $M = O(cn^2/p)$ memory per processor. Most commonly, 2D processor grids are used by algorithms that achieve this communication complexity for $c = 1$, but 3D processor grids and more complicated schemes are needed to achieve the complexity with any $c \in [1, p^{1/3}]$ and obtain practical performance improvements [35]. For Cholesky factorization, which is simpler than LU and QR, these algorithms attain communication lower bounds $W = O\left(\frac{n^3}{pM^{1/2}}\right)$ [7] and $W \cdot S = \Omega(n^2)$ [34], for a range of W parameterized by c.

The best previously known algorithms for solving the symmetric eigenvalue problem directly, use 2D parallelizations and achieve the cost $W = O(n^2/\sqrt{p})$. We introduce algorithms that reduce the horizontal communication cost asymptotically by a factor of $\sqrt{c}$, while using a factor of c more memory and $\sqrt{c}$ more synchronizations, in the same fashion as previously done for one-sided factorizations. The new algorithms generalize of previous approaches, and the flexibility offered by the parameter c increases the dimensionality of the tuning space for symmetric eigensolver implementations. In particular, employing a large c is attractive for bandwidth-constrained problems on massively-parallel architectures.

Our algorithms focus on reducing the symmetric matrix to thinner and thinner banded matrices with the same eigenvalues. This "successive band reduction" approach [10, 11], i.e. reducing to an

Grey Ballard was supported by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its U.S. Department of Energy Contract No. DE-AC04-94AL85000. James Demmel was supported by US Dept. of Energy, Office of Science, Office of Advanced Scientific Computing Research, Grant DOE DE-SC0010200, and ASPIRE Lab industrial sponsors and affiliationliates Intel, Google, Hewlett-Packard, Huawei, LGE, NVIDIA, Oracle, and Samsung. Edgar Solomonik was supported by a US. Dept. of Energy Computational Science Graduate Fellowship and an ETH Zurich Postdoctoral Fellowship.

SPAA '17, July 24-26, 2017, Washington DC, USA

ACM ISBN 978-1-4503-4593-4/17/07…$15.00
https://doi.org/10.1145/3087556.3087561

intermediate banded matrix rather than directly to tridiagonal, has been used to reduce vertical communication and synchronization costs [8]. Further, in practice, algorithms using a two-stage (full-to-banded and banded-to-tridiagonal) approach [5, 25] have been shown to outperform libraries that reduce directly to tridiagonal (like ScaLAPACK [12]). However, a disadvantage of successive band reduction is an increase in cost of the back transformations done to compute eigenvectors. Unlike the forward application of transformations whose computation cost scales linearly with the matrix band-width, known algorithms for back transformations require $O(n^3)$ operations for each intermediate band-width used.

The BSP model allows us to formulate and analyze algorithms as compositions of a set of common building-blocks. We leverage algorithms for matrix multiplication and QR factorization within our symmetric eigensolvers. For QR factorization, we extends known algorithms for tall-and-skinny matrices [16] and square matrices [38] to be efficient for arbitrary rectangular matrices.

We use these building blocks to define algorithms for reducing a dense matrix to a banded matrix, and a banded matrix to a thinner band-width, while preserving eigenvalues. Our main algorithm combines these, using $O(\log p)$ intermediate band-widths. The algorithm is work-efficient for computing eigenvalues, requires $O(n^2/\sqrt{cp})$ horizontal communication, $O(n^2 \log p/\sqrt{cp})$ vertical communication, and $O(\sqrt{cp}\log^2 p)$ synchronizations (BSP supersteps). Known approaches for back-transformations to compute eigenvectors require the same asymptotic amount of computation for matrices of any band-width, meaning our approach may require a computation cost of $O(n^2 k \log p/p)$ if k eigenvectors are needed. We leave the analysis of back-transformation computation for future work, but propose a potential approach to reduce the number of intermediate band-widths needed by our symmetric eigensolver.

2 THEORETICAL COST MODEL

We use the Bulk Synchronous Parallel (BSP) model [39] with an additional parameter to measure the cost of traffic between memory and cache. We derive asymptotic bounds on the parallel running-time of our algorithms for this two-level architectural model, with consideration for both communication between processors and in the memory hierarchy of each processor. The BSP model permits an all-to-all communication to be done with unit synchronization cost, which will allow us to construct BSP algorithms for general matrix distributions and compose them without significant overhead.

We employ cost notation typically used for the α–β communication model. As all stored and communicated datasets in this paper consist exclusively of floating-point numbers, we quantify sizes in terms of 'words' (floating-point numbers of a given precision). We model the memory hierarchy of each processor by a main 'slow' memory (i.e. DRAM) and a 'fast' memory (i.e. cache). We permit interprocessor (horizontal) communication to move data between main memories of different processors, and intraprocessor (vertical) communication to move data between main memory and cache of a single processor. Our architectural model is characterized by the following parameters:

- p – processors on a fully-connected network,
- M – words of memory owned by each processor,
- H – words of cache owned by each processor,
- γ – time to compute a floating point operation,
- β – time to send or receive a word,
- ν – time to move a word between cache and memory,
- α – time to perform a (global) synchronization.

We bound the cost of each algorithm by measuring four quantities:

- F – number of local floating point operations performed (computation cost),
- W – number of words of data moved between processors (horizontal communication cost),
- Q – number of words of data moved between main memory and cache (vertical communication cost),
- S – number of BSP supersteps (synchronization cost).

If at each superstep $i \in [1, S]$, processor j performs F_i^j local operations, sends and receives W_i^j total words, and performs Q_i^j reads and writes to memory, then the costs of the BSP algorithm are

$$F = \sum_{i=1}^{S} \max_{j\in[1,p]} F_i^j, \quad W = \sum_{i=1}^{S} \max_{j\in[1,p]} W_i^j, \quad Q = \sum_{i=1}^{S} \max_{j\in[1,p]} Q_i^j,$$

and the BSP execution time of this algorithm is

$$T = \Theta(\gamma \cdot F + \beta \cdot W + \nu \cdot Q + \alpha \cdot S).$$

This model does not consider overlap between communication and computation (or between other costs), as such overlap does not affect the overall asymptotic time.

We simplify asymptotic cost expressions by assuming $\gamma \leq \beta$. Further, we write only vertical communication terms which are not associated with horizontal communication or with computations that achieve a factor of $\sqrt{H}$ cache reuse (optimal for matrix multiplication [27]). These simplifications correspond to the assumptions on the relative communication times, $\nu \leq \beta$ and the floating point rate $\nu \leq \gamma \cdot \sqrt{H}$. However, general vertical communication cost upper-bounds may be obtained from our stated results for arbitrary ν by reinserting the term $O(\nu \cdot (F/\sqrt{H} + W))$.

We will provide asymptotic bounds for the BSP cost of all algorithms in the paper. Sometimes, we will employ algorithms as building blocks whose cost has been analyzed in the standard $\alpha - \beta$ model, which is restricted to point-to-point messaging (pairwise synchronization). These algorithms are trivially translated to the BSP model used in this paper, which is less restrictive (allows bulk synchronizations).

Throughout the paper, we will assume that matrix dimensions are greater than and divisible by the number of processors. When it is clear that the asymptotic costs would not be affected, we will also omit floors and ceilings when subdividing the number of processors and matrix dimensions.

3 BUILDING BLOCKS

We first state known results and provide minor extensions to quantify the complexity of matrix multiplication and of QR factorization in our cost model. These results will be critical in the cost analysis of the new symmetric eigensolvers, which use matrix multiplication and QR factorization as subroutines.

3.1 Matrix Multiplication

Our symmetric eigensolvers will perform matrix multiplications, often of nonsquare matrices. We consider the BSP cost of multiplication of arbitrary rectangular matrices with any starting distribution. Additionally, we specially consider the BSP cost of a matrix multiplication of a pre-replicated matrix with another matrix in an arbitrary distribution. We start with the vertical communication cost of a matrix multiplication done by a single processor.

Lemma 3.1. *The multiplication of matrices of dimensions $m \times n$ and $n \times k$ can be done by a single processor in time,*

$$O(\gamma \cdot mnk + \nu \cdot [mn + mk + nk]).$$

The Rec-Mult algorithm [22, Theorem 1] obtains the vertical communication cost given in Lemma 3.1. We omit the usual term $O(\nu \cdot mnk/\sqrt{H})$, since we have $\nu \leq \gamma \cdot \sqrt{H}$.

We now consider the full BSP cost of parallel rectangular matrix multiplication. The communication cost of square matrix multiplication is well known [1, 3, 9, 14, 29, 31]. The horizontal costs of rectangular matrix multiplication have also been analyzed within the α–β communication model, where a recursive algorithm was proposed [15] that attains the communication lower bound. We show that the algorithm in [15] can be executed within the time specified in the subsequent Lemma, for any initial load balanced distribution of the matrices. It is possible to also design different matrix multiplication algorithms in the BSP model with a $\Theta(\log p)$ factor less in synchronization cost, but the overall synchronization costs of our QR and symmetric eigensolve algorithms would not improve. We parameterize the memory used by the algorithm by a parameter v, which controls how many block matrix multiplications are performed.

Lemma 3.2. *For any $v \geq 1$, the multiplication of matrices of dimensions $m \times n$ and $n \times k$ in any load-balanced starting layout can be done in BSP time,*

$$O\Bigg(\gamma \cdot \frac{mnk}{p} + \beta \cdot \Bigg[\frac{mn + nk + mk}{p} + v^{1/3}\left(\frac{mnk}{p}\right)^{2/3}\Bigg] + \alpha \cdot v \log p\Bigg),$$

using $M = O\left(\frac{mn+nk+mk}{p} + \left(\frac{mnk}{vp}\right)^{2/3}\right)$ memory.

Proof. We consider the cost of the recursive 'CARMA' algorithm [15]. The algorithm assumes specific initial matrix layouts, but does not assume any initial data is replicated. Therefore, starting from load balanced layouts, the BSP time to move to the layouts specified by CARMA is $O(\beta \cdot \frac{mn+nk+mk}{p} + \alpha)$. Because the computation is load balanced, the computation cost is $O(\gamma \cdot mnk/p)$. The latency cost of the CARMA algorithm is an upper-bound on the number of BSP supersteps necessary to execute it. In [15], the latency cost is shown to be $O\left(\frac{mnk}{pM^{3/2}} \log p\right) = O(v \log p)$. The communication cost of CARMA is presented in cases for 1D, 2D, and 3D processor grids. We show that the postulated BSP time upper-bound holds for all cases.

We first argue that the vertical communication cost of the local matrix multiplications (given by Lemma 3.1) is dominated by horizontal communication due to the assumption $\beta \geq \nu$. In the 3D case, the operand matrix blocks are nearly square, and either one of the operands or the output is always communicated, so horizontal communication cost dominates vertical communication cost. In the 1D and 2D cases, each processor performs a single local matrix multiplication, where the largest operand has size $O(\frac{mn+nk+mk}{p})$, since it is the local block of the largest matrix.

We leverage previous analysis [15] to derive the horizontal communication cost. Let $d_1 = \min(m, n, k)$, $d_2 = \text{median}(m, n, k)$, and $d_3 = \max(m, n, k)$. If $p < d_3/d_2$ (1D case), then $d_1 d_2 < d_1 d_3/p$, so the provided cost $O(\beta \cdot d_1 d_2) = O(\beta \cdot (mn + nk + mk)/p)$. If $d_3/d_2 \leq p \leq d_2 d_3/d_1^2$ (2D case), then the provided cost $O(\beta \cdot \sqrt{d_1^2 d_2 d_3/p}) = O(\beta \cdot (mn + nk + mk)/p)$. Finally, if $p > d_2 d_3/d_1^2$ (3D case), the provided cost $O(\beta \cdot [mnk/(p\sqrt{M}) + (mnk/p)^{2/3}]) = O(\beta \cdot v^{1/3}(mnk/p)^{2/3})$. □

The algorithm analyzed in Lemma 3.2 allows any initial load balanced matrix distributions. We now consider Algorithm 3.1, which assumes an initial distribution with replicated data and subsequently can multiply certain matrices in less time than given by Lemma 3.2. In Algorithm 3.1, one of the input matrices is stored redundantly on $c = p^{2\delta-1}$ 2D processor grids for any $c \in [1, p^{1/3}]$ ($\delta \in [1/2, 2/3]$). The parameterization by δ is the same as α in [38], while c is the same replication factor as in [35]. The parameter w controls the number of supersteps in Algorithm 3.1.

The algorithm permits the distribution to be defined as a blocking of the matrices after permutation by $P^{(1)}, P^{(2)}$. Our analysis assumes the blocking is roughly, but not necessarily exactly load balanced, permitting both cyclic and block-cylic matrix factorization algorithms where different processors perform updates (matrix multiplications) with a slightly different amount of local data at each step. We will employ Algorithm 3.1 with cyclic distributions, for which $P^{(1)}_{ij} = 1$ for $i = (j \bmod q)(m/q) + \lfloor j/q \rfloor$ and $P^{(2)}_{jk} = 1$ for $k = (j \bmod q)(n/q) + \lfloor j/q \rfloor$. On each processor grid layer, the algorithm executes a variant of the SUMMA algorithm [40], which communicates the operand B and reduces the output C. This variant is chosen, since we will use the algorithm with the operand A being larger in dimensions than B and C.

Lemma 3.3. *Consider Algorithm 3.1 for multiplication of matrices A and B of dimensions $m \times n$ and $n \times k$, where the initial distributions of A and B satisfy the stated requirements for permutations $P^{(1)}$ and $P^{(2)}$ where each block A_{ij} of $P^{(1)}AP^{(2)}$ has dimensions $O(m/p^{1-\delta}) \times O(n/p^{1-\delta})$. Then, using $M = O(mn/p^{2(1-\delta)} + (mk + nk)/(wp^{\delta}))$ memory for any $w \in [1, p^{1-\delta}]$, the algorithm can be executed in BSP time,*

$$O\left(\gamma \cdot \frac{mnk}{p} + \beta \cdot \frac{mk + nk}{p^{\delta}} + \alpha \cdot w\right),$$

when $H \geq mn/p^{2(1-\delta)}$ and the copies of A start inside cache, and otherwise with an extra cost of $O(\nu \cdot \frac{wmn}{p^{2(1-\delta)}})$.

Proof. As required by Algorithm 3.1, B starts in any load-balanced distribution over the p processors. As the initial layout is load-balanced the redistribution done on line 4 costs $O(\beta \cdot nk/p + \alpha)$. The gather on line 9 and reduce-scatter on line 11 are dual communication patterns that together have cost $O(\beta \cdot (mk + nk)/(qcw) + \alpha)$.

Algorithm 3.1 $[C] \leftarrow$ Streaming-MM$(A, B, P^{(1)}, P^{(2)}, \Pi)$

Require: Given positive integers p, m, n, k, w and $\delta \in [1/2, 2/3]$: Π is a grid of $q \times q \times c$ processors with $q = p^{1-\delta}$ and $c = p^{2\delta-1}$, A is $m \times n$, B is $n \times k$. For each $l \in [1, c]$, $\Pi[i, j, l]$ owns all elements in A_{ij}, defined by square permutation matrices $P^{(1)}$, $P^{(2)}$, as $P^{(1)}AP^{(2)} = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & \ddots & \vdots \\ A_{q1} & \cdots & A_{qq} \end{bmatrix}$.

1: B is in any load balanced layout over all p processors.
2: Let $z = wc$
3: Partition B into blocks: $P^{(2)T}B = \begin{bmatrix} B_{11} & \cdots & B_{1z} \\ \vdots & & \vdots \\ B_{q1} & \cdots & B_{qz} \end{bmatrix}$.
4: Redistribute B so that each $\Pi[i, j, l]$ owns $k/(zq)$ columns of B_{jh} for each $h \in \{l, l + c, \ldots, l + (w - 1)c\}$.
5: *% Execute loop iterations in parallel*
6: **for** $i \in [1, q], j \in [1, q], l \in [1, c]$ **do**
7: *% Execute loop iterations in sequence*
8: **for** $h \in \{l, l + c, \ldots, l + (w - 1)c\}$ **do**
9: Gather B_{jh} on $\Pi[i, j, l]$
10: Compute $\bar{C}_{ijh} = A_{ij} \cdot B_{jh}$ on $\Pi[i, j, l]$
11: Reduce-scatter $C_{ih} = \sum_{j=1}^{c} \bar{C}_{ijh}$ so that each $\Pi[i, j, l]$ owns $k/(zq)$ columns of C_{ih}

Require: $C = A \cdot B$ is distributed so that each processor in Π owns mk/p elements of C.

Over all w iterations of index h, we get $O(\beta \cdot (mk+nk)/(qc)+\alpha \cdot w) = O(\beta \cdot (mk + nk)/p^{\delta} + \alpha \cdot w)$.

The w local matrix multiplications take time,

$$O\left(\gamma \cdot \frac{mnk}{p} + \nu \cdot \left(\frac{wmn}{p^{2(1-\delta)}} + \frac{mk + nk}{p^{\delta}}\right)\right),$$

by Lemma 3.1. However, if the entire matrix A starts in cache, which is possible if $H \geq mn/p^{2(1-\delta)}$, it suffices to read only the entries of B_{jh} from memory into cache and write the entries of $\bar{C}_{ijh}$ out to memory. In this case, the vertical communication cost is $O(\nu \cdot \frac{mk+nk}{qc}) = O(\nu \cdot \frac{mk+nk}{p^{\delta}})$. This term is dominated by the interprocessor communication term since $\beta \geq \nu$. The memory usage corresponds to the storage necessary for each block: A_{ij}, B_{jh}, and $\bar{C}_{ijh}$, $M = O\left(\frac{mn}{p^{2(1-\delta)}} + \frac{mk+nk}{wp^{\delta}}\right)$. □

3.2 QR Factorization

We will use QR factorization within our symmetric eigensolver algorithms to obtain orthogonal transformations that introduce zeros when applied to the symmetric matrix. The vertical communication cost of executing a sequential QR factorization is proportional to that of matrix multiplication.

Lemma 3.4. *The QR factorization of an $m \times n$ matrix A with $m \geq n$ can be done sequentially in time, $O(\gamma \cdot mn^2 + \nu \cdot mn)$.*

The sequential Communication-Avoiding QR (CAQR) algorithm achieves the vertical communication cost given above [16]. The Householder representation, lower trapezoidal $m \times n$ matrix U and upper-triangular $n \times n$ matrix T so that $Q = I - UTU^T$, may be obtained with the cost of Lemma 3.4 using Householder reconstruction [6].

We first consider parallel QR factorization of square matrices.

Lemma 3.5. *The QR factorization of an $n \times n$ matrix A distributed in any load-balanced layout can be computed using $M = O\left(\frac{n^2}{p^{2(1-\delta)}}\right)$ memory for any $\delta \in [1/2, 2/3]$ in BSP time,*

$$O\left(\gamma \cdot \frac{n^3}{p} + \beta \cdot \frac{n^2}{p^{\delta}} + \alpha \cdot p^{\delta}\right).$$

The QR algorithm given by [38] in the BSP model achieves the costs given in Lemma 3.5. The vertical communication cost was not analyzed in [38]. However, the algorithm consists purely of distributed matrix multiplications or QR factorizations, which by Lemma 3.1 and Lemma 3.4 have a vertical communication cost proportional to the matrix sizes. As the analysis in [38] assumes all matrices that participate in multiplication or QR factorization are communicated, due to $\nu < \beta$, the horizontal communication cost dominates the vertical communication costs associated with these operations.

We now adapt the QR algorithm from [38] to handle rectangular matrices with a desirable asymptotic cost (the embedding used in [38] is inefficient for tall-and-skinny matrices). Our adaptation is based on a binary QR reduction tree, with QR factorizations of nearly square matrices done at every node in the tree performed using the algorithm from [38]. An approach employing a QR reduction tree using Givens rotations goes back to [23], a blocked flat tree approach (optimal sequentially) was presented in [24], and a parallel block reduction tree approach was given earlier in [13]. Our approach is closest to the TSQR algorithm [16], except a set of up to $q_{\max}$ processors works on each tree node.

Algorithm 3.2 computes the QR factorization of an $m \times n$ matrix, outputting the first n columns of the orthogonal Q factor, as well as the $n \times n$ upper-triangular matrix R. The algorithm assumes the existence of a sequential routine 'QR' and a parallel routine for (nearly) square matrices 'square-QR'.

Theorem 3.6. *Algorithm 3.2 can compute the QR factorization of any $m \times n$ matrix A with $m \geq n$ in a load-balanced layout, using $M = O\left(\left(\frac{n^{\delta} m^{1-\delta}}{p^{1-\delta}}\right)^2\right)$ memory for any $\delta \in [1/2, 2/3]$, in BSP time,*

$$O\left(\gamma \cdot \frac{mn^2}{p} + \beta \cdot \left(\frac{m^{\delta} n^{2-\delta}}{p^{\delta}} + \frac{mn}{p}\right) + \alpha \cdot \left(\frac{np}{m}\right)^{\delta} \log^2 p\right).$$

Proof. We assume without loss of generality that m/n and p are powers of two. Let $T(\bar{m})$ be the cost of Algorithm 3.2 for an $\bar{m} \times n$ matrix using p processors. Note that m corresponds to the number of rows in the original input matrix, while $\bar{m}$ will be used to refer to the number of rows at a given recursive step. We select the maximum number of processors to be used in base-case square QR factorizations to be $q_{\max} = \frac{pn}{m} \log(p)^{1/\delta}$, in order to minimize synchronization cost while achieving an optimal horizontal communication cost.

The cost of the sequential base case of Algorithm 3.2 is, by Lemma 3.4, $T_{\text{bs1}}(\bar{m}) = O(\gamma \cdot \bar{m}n^2 + \nu \cdot \bar{m}n)$. When reaching the square base case (dimension $2n \times n$, since m/n is a power of two), we employ the square QR algorithm [38] with up to $q_{\max} = \frac{pn}{m} \log(p)^{1/\delta}$

Algorithm 3.2 $[Q, R] \leftarrow$ rect-QR(A, Π)

Require: Given positive integers $p, m, n, q_{\max}$ and $\delta \in [1/2, 2/3]$: Π is a set of p processors, A is $m \times n$, m/n and p are powers of two, and each $\Pi[i]$ owns mn/p elements of A.

1: **if** $p = 1$ **then** Compute $[Q, R] =$ QR(A) sequentially and exit.
2: **if** $m \leq 2n$ **then** Compute $[Q, R] =$ square-QR$(A, \Pi[1 : \min(p, q_{\max})])$ and exit.
3: Let $r = \min(p, \lceil \frac{m}{2n} \rceil)$ and partition $A = \begin{bmatrix} A_1^T & \cdots & A_r^T \end{bmatrix}^T$ so that each A_i is $m/r \times n$
4: *% Execute loop iterations in parallel*
5: **for** $i \in [1, r]$ **do**
6: $\quad [W_i, R_i] =$ rect-QR$(A_i, \Pi[(i-1)(p/r)+1 : i(p/r)])$
7: $[Z, R] =$ rect-QR$\left(\begin{bmatrix} R_1^T & \cdots & R_r^T \end{bmatrix}^T, \Pi\right)$
8: Partition $Z = \begin{bmatrix} Z_1^T & \cdots & Z_r^T \end{bmatrix}^T$ so that each Z_i is $n \times n$
9: *% Execute loop iterations in parallel*
10: **for** $i \in [1, r]$ **do**
11: $\quad$ Compute $Q_i = W_i Z_i$ using $\Pi[(i-1)(p/r) + 1 : i(p/r)]$

Require: $A = Q \cdot R$ where $Q = \begin{bmatrix} Q_1^T & \cdots & Q_r^T \end{bmatrix}^T$ is $m \times n$ with orthogonal columns, R is $n \times n$ and upper-triangular, both are distributed in load balanced layouts across Π.

processors. We can bound the cost of this QR by Lemma 3.5. We break the cost into two cases: $T_{\text{bp}}(\bar{p}) = T_{\text{bp1}}(\bar{p})$ when $\bar{p} < q_{\max}$ and $T_{\text{bp}}(\bar{p}) = T_{\text{bp2}}$ when $\bar{p} \geq q_{\max}$, where

$$T_{\text{bp1}}(\bar{p}) = O(\gamma \cdot n^3/\bar{p} + \beta \cdot n^2/\bar{p}^{\delta} + \alpha \cdot \bar{p}^{\delta}),$$

$$T_{\text{bp2}} = O\left(\gamma \cdot \frac{mn^2}{p \log(p)^{1/\delta}} + \beta \cdot \frac{m^{\delta} n^{2-\delta}}{p^{\delta} \log p} + \alpha \cdot \left(\frac{np}{m}\right)^{\delta} \log p\right).$$

The square QR algorithm requires that the matrix be embedded into a slanted panel [38]. This can be done generally by using a somewhat larger matrix, but in all except the first recursive call, the $2b \times b$ matrix will have the structure of two stacked upper-triangular matrices. The rows of these upper-triangular matrices can be interleaved to produce a slanted panel without embedding into a larger matrix.

The recursive calls on line 6 always immediately encounter one of the base-cases. The only time base cases can have a matrix with dimension other than $2n \times n$ is during the invocations on line 6 at the first recursive step of the algorithm, and only when $m > 2np$. Therefore, we consider this first recursive step of Algorithm 3.2 separately. The cost of the first recursive step, when $m > 2np$, includes

- the cost of a potential redistribution, $O(\beta \cdot mn/p + \alpha)$,
- the cost of the invocations on line 6 (which lead to base cases), $T_{\text{bs1}}(m/p)$, since $r = \min(p, \lceil m/2n \rceil) = p$,
- the cost of the matrix multiplications on line 11, which are done concurrently, each by a single processor, is $O(\gamma \cdot mn^2/p + \nu \cdot mn/p)$.

Therefore, the total BSP time of the algorithm for $m > 2np$ is

$$T(m) = T(np) + T_{\text{bs1}}\left(\frac{m}{p}\right) + O\left(\gamma \cdot \frac{mn^2}{p} + \beta \cdot \frac{mn}{p}\right)$$
$$= T(np) + O(\gamma \cdot mn^2/p + \beta \cdot mn/p + \alpha).$$

The cost of this initiale step for $m > 2np$ is no greater than the cost in the theorem. Subsequent recursive calls into line 11 or the case when $m \leq 2np$, the matrix multiplications done on line 11 involve matrices of size at most $2n \times n$, each executed using $pn/\bar{m}$ processors. By Lemma 3.2 with $\upsilon = (pn/\bar{m})^{2-3\delta}$, these matrix multiplications (done concurrently) take time, $T_{\text{MM}}(\bar{m}) =$

$$O\left(\gamma \cdot \frac{\bar{m}n^2}{p} + \beta \cdot \left(\frac{\bar{m}n}{p} + \frac{\bar{m}^{\delta} n^{2-\delta}}{p^{\delta}}\right) + \alpha \cdot \left(\frac{pn}{\bar{m}}\right)^{2-3\delta} \log p\right)$$

and use $M = O\left(\left(\frac{n^{\delta} \bar{m}^{1-\delta}}{p^{1-\delta}} \log p\right)^2\right)$ memory. When combined with the concurrent recursive calls on line 11 on matrices of size $2n \times n$ with $pn/\bar{m}$ processors and the recursive call on line 7 on a matrix of size $\bar{m}/2 \times n$ with all p processors, we obtain the following BSP time recurrence for $\bar{m} \leq 2np$,

$$T(\bar{m}) = T(\bar{m}/2) + T_{\text{bp}}(pn/\bar{m}) + T_{\text{MM}}(\bar{m}),$$

where $T_b(pn/\bar{m})$ is a base case where up to $q_{\max}$ processors perform the QR. We consider the two cases (for $\bar{m} \leq 2np$),

$$T(\bar{m}) = T(\bar{m}/2) + T_{\text{MM}}(\bar{m}) + \begin{cases} T_{\text{bp1}}(pn/\bar{m}) & : pn/\bar{m} < q_{\max} \\ T_{\text{bp2}} & : pn/\bar{m} \geq q_{\max} \end{cases}$$

Since $q_{\max} = \frac{pn}{m} \log(p)^{1/\delta}$, and $\bar{m}$ decreases by a factor of two at each step, up to the first $(1/\delta) \log \log p$ recursive steps make the call on line 6 with more than $q_{\max}$ processors. The computation and communication cost of these calls are no greater than that of matrix multiplication (part of $T_{\text{MM}}(\bar{m})$), while the synchronization cost increases geometrically, up to the latency cost in T_{bp2}. Therefore, the recurrence is asymptotically equivalent to (for $\bar{m} \leq 2np$),

$$T(\bar{m}) = T(\bar{m}/2) + T_{\text{MM}}(\bar{m}) + T_{\text{bp2}}$$
$$= T(\bar{m}/2) + O\left(\gamma \cdot \left(\frac{\bar{m}n^2}{p} + \frac{mn^2}{p \log(p)^{1/\delta}}\right)\right.$$
$$\left. + \beta \cdot \left(\frac{\bar{m}n}{p} + \frac{\bar{m}^{\delta} n^{2-\delta}}{p^{\delta}} + \frac{m^{\delta} n^{2-\delta}}{p^{\delta} \log p}\right) + \alpha \cdot \left(\frac{pn}{m}\right)^{\delta} \log p\right).$$

Since, $\bar{m} \leq 2np$, one of the base-cases is reached after $\log p$ steps, and so the above time is the one postulated in the theorem. □

Alternate communication-efficient formulations of a rectangular QR algorithm are also possible (for instance by combining column-recursion [20] with communication-efficient matrix multiplication, see [32]). We would like to work with the Householder representation to apply orthogonal transformations efficiently in our symmetric eigensolver algorithms, so we give the following corollary.

Corollary 3.7. *The Householder representation of the $m \times n$ orthogonal matrix Q computed by Algorithm 3.2, $Q = (I - UTU_1^T)$, where U_1 is the lower triangular top $n \times n$ block of U, while T is upper-triangular and $U^T U = T^{-1} + T^{-T}$, can be obtained with the same cost and memory usage as in Theorem 3.6.*

Proof. The Householder representation U, T can be obtained stably by executing $[U_1, W_1] = \text{LU}(Q_1 - S)$ where Q_1 is the top $n \times n$ block of Q and S is a diagonal sign matrix, then computing $U = QW_1^{-1}$ and $T = W_1 U_1^{-T}$ [6]. The matrices U_1, W_1, U_1^{-1}, and W_1^{-1} can be obtained by a parallel non-pivoted LU factorization algorithm augmented to subtract S as in [6], which makes the

matrix diagonally dominant. The LU algorithms in [37] and [35] both obtain the desired costs. We use the former in our analysis.

When executed using pn/m processors, the algorithm in [37] takes BSP time, $O(\gamma \cdot mn^2/p + \beta \cdot m^\delta n^{2-\delta}/p^\delta + \alpha \cdot (np/m)^\delta)$. This cost was presented in [37], modulo analysis of vertical communication cost, but as the algorithm is based purely on parallel multiplication of square matrices, the vertical communication cost is dominated by the horizontal communication cost. The algorithm also outputs the inverses of the triangular factors [37], so matrix multiplications suffice to compute $U = QW_1^{-1}$ and $T = W_1 U_1^{-T}$. These can be done using all the processors in time, $O(\gamma \cdot mn^2/p + \beta \cdot m^\delta n^{2-\delta}/p^\delta + \alpha)$ with $M = O\left(\left(\frac{n^\delta m^{1-\delta}}{p^{1-\delta}}\right)^2\right)$ memory. As these costs and memory usage are no greater than in Theorem 3.6, we arrive at the postulated conclusion. □

4 SYMMETRIC EIGENSOLVERS

Algorithms for blocked computation of the eigenvalue decomposition of a symmetric matrix via a tridiagonal matrix were studied by [18, 19, 28]. These algorithms reduce an $n \times n$ symmetric matrix A to a matrix B with band-width b and the same eigenvalues as A via a series of $k = (n-b)/b$ orthogonal transformations,

$$B = Q_1^T \cdots Q_k^T A Q_k \cdots Q_1,$$

where each Q_i is representable in terms of b Householder vectors, aggregated in a trapezoidal matrix U_i, as $Q_i = (I - U_i T_i U_i^T)$.

A key property employed by these algorithms is that each two-sided trailing matrix update of blocked Householder transformations may be done as a rank-$2b$ symmetric update. To compute the two-sided transformation $Q^T X Q$ where $X = X^T$ and $Q = (I - UTU^T)$, we can write

$$\begin{aligned} Q^T X Q =& (I - UT^T U^T) X (I - UTU^T) \\ =& X + UV^T + VU^T, \end{aligned} \tag{4.1}$$

where $V = \frac{1}{2} UT^T U^T X U T - XUT$. This form of the update is cheaper to compute than the explicit two-sided update and is easy to aggregate by appending additional vectors to U (to aggregate the Householder form itself requires computing a larger T matrix). Since the trailing matrix update does not have to be applied immediately, but only to the columns which are factorized, this two-sided update can also be aggregated and used in a left-looking algorithm. For instance, to multiply $Q^T X Q$ by a matrix Y, we can compute

$$Q^T X Q Y = XY + UV^T Y + VU^T Y. \tag{4.2}$$

Returning to algorithms that compute a series of k two-sided transformations, we note that when computing V_2 from U_2 (to apply Q_2), we need to multiply U_2 by a submatrix of $Q_1^T A Q_1$, which can be done without applying Q_1, using the above form. Left-looking algorithms which generalize this idea and employ a delayed trailing matrix update have been used to reduce directly to tridiagonal form ($b = 1$) [18].

However, there are disadvantages to reducing the symmetric matrix directly to tridiagonal form, since it requires that a vector be multiplied by the trailing matrix for each computation of V_i of which there are $n-2$. These matrix-vector multiplications require $O(n)$ synchronizations and $O(n)$ transfers of the trailing matrix between memory and cache (so long as it does not fit into cache).

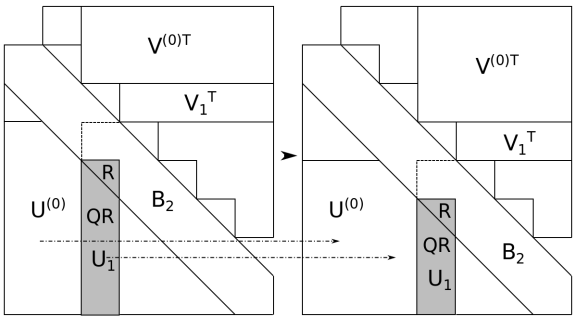

Figure 1: A depiction of matrices used in Algorithm 4.1 for two subsequent recursive steps.

These disadvantages motivated approaches where the matrix is not reduced directly to tridiagonal form, but rather to banded form, which allows for $b > 1$ Householder vectors to be computed via QR at each step without needing to touch the trailing matrix from within the QR. After such a reduction to banded form, it is then necessary to reduce the banded matrix to tridiagonal form. However, this can be significantly less expensive because the trailing matrix is banded and requires less work and vertical communication to update than during the full-to-banded reduction step.

Such a multi-stage reduction approach was introduced by [10, 11] with the aim of achieving BLAS 3 reuse. These algorithms can reduce the banded matrix to tridiagonal or perform more stages of reduction, employing multiple intermediate band-widths. Performing more stages of successive band reduction can improve the synchronization cost of the overall approach, from $O(n)$ as needed if reducing to tridiagonal form directly, to $O(\sqrt{p})$ as shown by [8]. ELPA [5] is a distributed-memory library implementing a two-step reduction approach, motivated by reducing vertical communication cost. ELPA employs the parallel banded-to-tridiagonal algorithm introduced by [30]. Performance studies by [5] have demonstrated that this approach is particularly beneficial for large matrices.

We first introduce an algorithm for reducing a full dense matrix to banded form, with up to $O(p^{1/6})$ less horizontal communication than previously known schemes. We subsequently introduce an algorithm for reducing a banded matrix to a smaller band-width, again with less communication than known approaches. Both of these reduction algorithms use a parallel routine 'QR', which performs QR factorization and outputs the Householder representation (U, T) of the Q factor. We then give a combined, 2.5D symmetric eigensolver algorithm, that uses the first algorithm to reduce the dense symmetric matrix to band-width $\frac{n}{\max(p^{2-3\delta}, \log p)}$, then uses $O(\log p)$ calls to our band-to-band reduction, to arrive at a band-width of n/p, which is small enough to allow for efficient sequential computation of eigenvalues. The resulting symmetric eigensolver has the same BSP complexity as QR factorization (Lemma 3.5), modulo logarithmic factors in the number of processors for the vertical communication and synchronization costs.

Algorithm 4.1 $[B] \leftarrow$ 2.5D-Full-to-Band$(A, U^{(0)}, V^{(0)}, \Pi, b)$

Require: Given nonnegative integers p, n, m, b and $\delta \in [1/2, 2/3]$, $z = (bp^\delta/n)^{(1-\delta)/\delta}$: Π is a grid of $q \times q \times c$ processors where $q = p^{1-\delta}$ and $c = p^{2\delta-1}$ and $b \bmod q = 1$, A is an n-by-n symmetric matrix, $U^{(0)}$ and $V^{(0)}$ are n-by-m matrices where $U^{(0)}$ is trapezoidal (zero in top right upper b-by-b triangle) and $V^{(0)}$ is dense, A (stored as a nonsymmetric matrix), $U^{(0)}$, and $V^{(0)}$ are distributed cyclically over $\Pi[:, :, k]$ for each $k \in [1, c]$.

1: **if** $n \leq b$ **then**
2: Compute $B = A + U^{(0)}V^{(0)^T} + V^{(0)}U^{(0)^T}$ and exit.
3: Subdivide $A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}$ where A_{11} is b-by-b
4: Subdivide $U^{(0)} = \begin{bmatrix} U_1^{(0)} \\ U_2^{(0)} \end{bmatrix}$ and $V^{(0)} = \begin{bmatrix} V_1^{(0)} \\ V_2^{(0)} \end{bmatrix}$ where $U_1^{(0)}$ and $V_1^{(0)}$ are b-by-m
5: Compute $\begin{bmatrix} \bar{A}_{11} \\ \bar{A}_{21} \end{bmatrix} = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} + U^{(0)}V_1^{(0)^T} + V^{(0)}U_1^{(0)^T}$
6: *% Compute QR of matrix panel*
7: $[U_1, T, R] \leftarrow \mathrm{QR}(\bar{A}_{21}, \Pi[:, 1:z, :])$
8: Compute $W = A_{22}U_1 + U_2^{(0)}(V_2^{(0)^T}U_1) + V_2^{(0)}(U_2^{(0)^T}U_1)$
9: Compute $V_1 = \frac{1}{2}U_1(T^T(U^T(WT))) - WT$
10: Replicate U_1 and V_1 so that they are distributed cyclically over $\Pi[:, :, k]$ for each $k \in [1, c]$
11: *% Recursively reduce the trailing matrix to banded form*
12: $B_2 = $ 2.5D-Full-to-Band$(A_{22}, [U_2^{(0)}, U_1], [V_2^{(0)}, V_1], \Pi, b)$
13: $B = \left[\begin{array}{c|cc} \bar{A}_{11} & R^T & 0 \\ \hline R & \multicolumn{2}{c}{\multirow{2}{*}{B_2}} \\ 0 & & \end{array}\right]$

Ensure: B is a symmetric n-by-n matrix with band-width b and the same eigenvalues as $A + U^{(0)}V^{(0)^T} + V^{(0)}U^{(0)^T}$.

4.1 Full-to-Band Reduction

Algorithm 4.1 reduces a symmetric n-by-n matrix A to band-width b using replication of data and aggregation. It achieves a horizontal communication cost of $W = O(n^2/p^\delta)$, when the amount of available memory on each processor is $M = O(n^2/p^{2(1-\delta)})$. The algorithm is left looking, meaning it updates the next matrix panel (line 5) immediately prior to performing the QR of the panel. Figure 1 displays the key matrices employed in Algorithm 4.1, specifically the third and fourth steps of recursion.

The algorithm replicates the matrix A and aggregates as well as replicates the updates $U^{(0)}$ and $V^{(0)}$ (these update matrices should have $m = 0$ columns for the initial invocation of Algorithm 4.1) over $c = p^{2\delta-1}$ layers of $q^2 = p^{2(1-\delta)}$ processors. In the definition of the algorithm and the analysis we assume that c and q are integers for any given p. Each of these replicated matrices is stored in a 2D cyclic distribution on each processor grid layer, adhering to the layout assumptions of Algorithm 3.1. A cyclic layout yields local blocks which can be used within sequential routines the same way as done in a blocked layout. The assumption $b \bmod q = 1$ ensures that whenever each new panel of U and V is replicated (U_1 and V_1 on line 10), they can be concatenated to previously replicated panels while maintaining a perfectly load balanced cyclic distribution.

Algorithm 4.1 performs the update correctly since, first, the computation of $W = \bar{A}U$ where $\bar{A} = Q^TAQ$ (line 8) follows the identity Eqn. (4.2). Further, as computed on line 9, V takes the desired form,

$$V = \left(\frac{1}{2}UT^TU^T - I\right)WT = \frac{1}{2}UT^TU^T\bar{A}UT - \bar{A}UT,$$

the same one as the aggregated update matrix derived in Eqn. (4.1). Consequently, the eigenvalues of the original matrix are preserved in the resulting banded matrix due to the ensured condition on the result of the tail recursion, which performs the update and factorization of the trailing matrix. In the base case, the matrix dimension is less than or equal to the desired matrix band-width, which means it suffices to perform the aggregated update and return the result, which would appear in the lower right block of the full banded matrix. We now analyze the execution time of Algorithm 4.1.

Lemma 4.1. *Algorithm 4.1 can reduce any symmetric n-by-n matrix (input in any evenly-distributed layout and with $n \geq p$) to a banded matrix with the same eigenvalues and any band-width $n/p^\delta \leq b \leq n/\log p$, using $M = O(n^2/p^{2(1-\delta)})$ memory for any $\delta \in [1/2, 2/3]$, when $H > 3n^2/p^{2(1-\delta)}$, in BSP time,*

$$O\left(\gamma \cdot \frac{n^3}{p} + \beta \cdot \frac{n^2}{p^\delta} + \alpha \cdot p^\delta \log^2 p\right).$$

If $H \leq 3n^2/p^{2(1-\delta)}$, then there is an additional vertical communication cost of $O(\nu \cdot (n/b)n^2/p^{2(1-\delta)})$.

Proof. Since $b \geq n/p^\delta$, we assume without loss of generality that $b \bmod p^{1-\delta} = 0$. We also note that since $b \geq n/p^\delta$, $z = (bp^\delta/n)^{(1-\delta)/\delta} \geq 1$. We note that the dimensions of A, $U^{(0)}$, and $V^{(0)}$ at any recursive step will always be less than the dimension of the original matrix, n. Algorithm 4.1 assumes A, $U^{(0)}$, and $V^{(0)}$ are initially replicated. Since each $b \times b$ block of these matrices is distributed cyclically and since $b \bmod q = 0$ ($q = p^{1-\delta}$), the submatrix extraction and concatenation done between recursive steps, can preserve perfect load balance without communication. To satisfy initial assumptions of the first invocation of Algorithm 4.1, we need to replicate the A matrix. Since, by assumption, it is distributed over all processors initially, the replication can be done with $O(n^2/q^2) = O(n^2/p^{2(1-\delta)})$ horizontal communication cost.

At each recursive step, Algorithm 4.1 performs a QR factorization, several matrix multiplications, and replicates U_1 and V_1. Each $O(n) \times b$ QR factorization is done using a processor subgrid of dimensions $p^{1-\delta} \times z \times p^{2\delta-1}$ with a total of $zp^\delta = p(b/n)^{(1-\delta)/\delta}$ processors (picked to minimize both communication and synchronization) using Algorithm 3.2. By Theorem 3.6 and the fact that $z \geq 1$, it takes BSP time,

$$O\left(\gamma \cdot \frac{n^{1/\delta}b^{3-1/\delta}}{p} + \beta \cdot \frac{nb}{p^\delta} + \alpha \cdot \frac{b}{n}p^\delta \log^2 p\right),$$

using $M = O\left(\left(\frac{b^\delta n^{1-\delta}}{(zp^\delta)^{1-\delta}}\right)^2\right) = O\left(\left(\frac{n(b/n)^{(2\delta-1)/\delta}}{p^{1-\delta}}\right)^2\right)$ memory.

The two matrix multiplications on line 5 and the five matrix multiplications on line 8 (done right to left), all correspond to an $O(n) \times O(n)$ replicated matrix multiplied by an $O(n) \times b$ rectangular

matrix. By Lemma 3.3, with $w = \max(1, bp^{2-3\delta}/n)$, using $M = O(n^2/p^{2(1-\delta)} + nb/(wp^\delta)) = O(n^2/p^{2(1-\delta)})$ memory, the time to compute these matrix multiplications is, if $U^{(0)}$ and $V^{(0)}$ start in cache,

$$O\left(\gamma \cdot \frac{n^2 b}{p} + \beta \cdot \frac{nb}{p^\delta} + \alpha \cdot w\right).$$

In general (for any cache size), there is an additional cost of $O(\nu \cdot \frac{(n/b)n^2}{p^{2(1-\delta)}})$. The memory usage needed for these matrix multiplications is greater than that needed for the QR factorizations done by each set of processors. Since $n^{1/\delta}b^{3-1/\delta} < n^2 b$ the computation cost of these matrix multiplications also dominates that of the QR factorizations.

The matrix multiplications needed to compute line 9 from right to left either operate on an $O(n) \times b$ matrix and a $b \times b$ matrix, like $W \cdot T$, or result in a $b \times b$ matrix, like $U^T \cdot (WT)$. By Lemma 3.2 any matrix multiplication where two of the matrix dimensions are b and one is $O(n)$, with $v = p^{2-3\delta}$, takes BSP time,

$$O\left(\gamma \cdot \frac{nb^2}{p} + \beta \cdot \left[\frac{nb}{p} + \frac{n^{2/3}b^{4/3}}{p^\delta}\right] + \alpha \cdot p^{2-3\delta}\log p\right).$$

Since $b \le n/\log p$, the above communication cost is never greater than that of the larger matrix multiplications, i.e. $n^{2/3}b^{4/3}/p^\delta \le nb/p^\delta$. The synchronization cost of the QR factorizations dominates that of of the matrix multiplications.

Replicating U_1 and V_1 over c subsets of q^2 processors (line 10) can be done in time, $O\left(\beta \cdot nb/p^{2(1-\delta)} + \alpha\right)$.

Therefore, the cost over all $n/b - 1$ recursive steps when all replicated matrices fit into cache (when $H > 3n^2/p^{2(1-\delta)}$) is the total cost postulated in the theorem. In the second scenario (when $H < 3n^2/p^{2(1-\delta)}$), the algorithm incurs an extra additive factor of $O\left((n/b)\frac{wn^2}{p^{2(1-\delta)}}\right)$ in vertical communication cost. The memory usage is dominated by the replicated matrix multiplication (invocation of Lemma 3.3 above), which is also as stated in the theorem. □

4.2 Band-to-Band Reduction

We now consider algorithms for reducing a banded matrix to a smaller band-width, while preserving eigenvalues. We start by recalling a parallel algorithm designed for small band-widths [8], then present Algorithm 4.2, which is designed to exploit additional parallelism given larger starting band-widths. Algorithm 4.2 describes the QR factorizations and applications necessary to reduce a symmetric banded matrix A from band-width b to band-width $h = b/k$ via bulge chasing. The algorithm eliminates n/h trapezoidal panels via QR factorization, each of which generate bulges of nonzeros in the trailing matrix. Each bulge is subsequently chased down the band by $O(n/b)$ eliminations again done by QR factorizations. Every new panel elimination is done immediately after the previously generated bulge is chased twice (including its initial panel elimination). Figure 2 depicts the QR factorizations necessary to eliminate a trapezoidal panel and chase two bulges generated from eliminating the first two panels, which are done concurrently in the algorithm. This type of pipelined successive band reduction approach was first considered by [10, 11]. The CA-SBR algorithm in [8] is similar, but assigns each processor a set of bulge chases at each pipeline step, rather than performing each bulge chase with a set of processors as done in Algorithm 4.2.

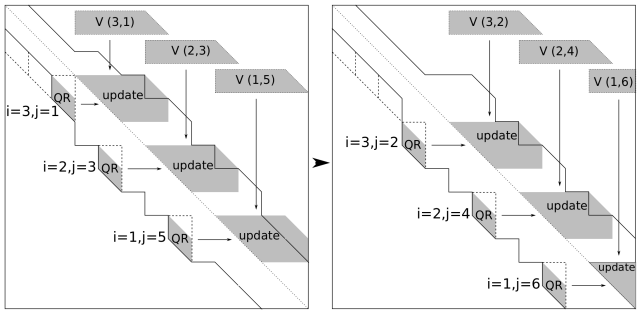

Figure 2: QR factorizations and updates in iterations $(i, j) \in \{(3,1),(2,3),(1,5)\}$ (left) and $(i,j) \in \{(3,2),(2,4),(1,6)\}$ (right) of Algorithm 4.1 with $k = 2$. These two sets of iterations are executed concurrently by processor groups $\hat{\Pi}_1$, $\hat{\Pi}_3$, and $\hat{\Pi}_5$ (left) and $\hat{\Pi}_2$, $\hat{\Pi}_4$, and $\hat{\Pi}_6$ (right), respectively. Only the unique part of the trailing matrix update is shown, while the pseudocode performs both symmetric reflections of it. Each matrix V is labeled with the iteration in which it is computed.

Lemma 4.2. *An $n \times n$ symmetric matrix (input in any load-balanced layout) of band-width $b \le n/p$ can be reduced to one with the same eigenvalues and band-width $b/2$, using $M = O(nb/p)$ memory, in BSP time,*

$$O\left(\gamma \cdot \frac{n^2 b}{p} + \beta \cdot nb + \nu \cdot \frac{n^2}{p} + \alpha \cdot p\right).$$

Proof. We consider the cost of one step of the CA-SBR algorithm [8]. A redistribution from any initial layout costs $O(\beta \cdot nb + \alpha)$. The analysis in [8] shows that the cost of reducing from bandwidth b to $b/2$ has the computation, horizontal communication, and synchronization costs, as well as the memory usage postulated in the lemma. The algorithm consits of a bulge chase pipeline, executed in $O(p)$ parallel steps, in which each processor works on $O(n/p)$ columns, chasing $O(n/(pb))$ bulges $O(n/(pb))$ times, for a total of $O(n^2/(p^2b^2))$ bulge chases. Since each bulge chase consists of a QR factorization and a matrix multiplication, with matrices of size $O(b) \times O(b)$, by Lemma 3.1 and Lemma 3.4, the vertical communication cost is $O(\nu \cdot b^2)$ for each bulge chase. Summing the costs of the bulge chases over all parallel steps yields the postulated total cost. □

We now consider the cost of Algorithm 4.2. Its primary innovation is to perform each QR factorization and update in parallel using a subset of processors, leveraging both pipelined parallelism across different bulge chases as well as parallelism within a bulge chase. When the band-width becomes smaller, fewer processors are used to execute each bulge chase.

Lemma 4.3. *Algorithm 4.2 can reduce an $n \times n$ symmetric matrix (input in any evenly-distributed layout) of band-width $b \ge n/p$ to one with the same eigenvalues and band-width b/k, using $M = O((n^{1-\delta}b^\delta/p^{1-\delta})^2)$ memory for any $\delta \in [1/2, 2/3]$ and any*

Algorithm 4.2 $[B] \leftarrow$ 2.5D-Band-to-Band(A, Π, b, k)

Require: Given positive integers b, p, n, k and $h = b/k$ with n mod $b \equiv 0$ and b mod $k \equiv 0$: A is a banded symmetric matrix of dimension n with band-width $b \leq n$, $\hat{\Pi}_j \subset \Pi$ is the jth group of $\hat{p} \equiv pb/n$ processors for $j \in [1, n/b]$.

1: Set $B = A$
2: Let $B[(j-1)b+1 : jb\ ,\ (j-1)b+1 : jb]$ be replicated in $\hat{\Pi}_j$ over $(bp/n)^{2\delta-1}$ subsets of $(bp/n)^{2(1-\delta)}$ processors.
3: *% Iterate over panels of B*
4: **for** $i \in [1, n/h-1]$ **do**
5: *% $\hat{\Pi}_j$ applies chase j of bulge i as soon as $\hat{\Pi}_{j-1}$ executes chase $(j-1)$*
6: **for** $j = 1 : \lfloor (n-ih-1)/b \rfloor$ **do**
7: *% Define row and column offsets*
8: Let $o_{\text{blg}} = (i-1)h + (j-1)b$, $o_{\text{qr.r}} = o_{\text{blg}} + h$
9: **if** $j = 1$ **then** $o_{\text{qr.c}} = o_{\text{qr.r}} - h$, $o_{\text{v}} = 0$
10: **else** $o_{\text{qr.c}} = o_{\text{qr.r}} - b$, $o_{\text{v}} = b - h$, $o_{\text{up.c}} = o_{\text{qr.c}} + h$
11: *% Define index ranges needed for bulge chase*
12: $n_{\text{r}} = \min(n - o_{\text{qr.r}}, b), n_{\text{c}} = \min(n - o_{\text{up.c}}, h + 3b)$
13: $I_{\text{qr.rs}} = o_{\text{qr.r}} + (1 : n_{\text{r}})$, $I_{\text{qr.cs}} = o_{\text{qr.c}} + (1 : h)$
14: $I_{\text{v.rs}} = o_{\text{v}} + (1 : n_{\text{r}})$, $I_{\text{up.cs}} = o_{\text{up.c}} + (1 : n_{\text{c}})$
15: *% Perform a rectangular parallel QR factorization*
16: $[U, T, R] \leftarrow \text{QR}(B[I_{\text{qr.rs}}, I_{\text{qr.cs}}], \hat{\Pi}_j[1 : ph/n])$
17: $B[I_{\text{qr.rs}}, I_{\text{qr.cs}}] = \begin{bmatrix} R \\ 0 \end{bmatrix}$, $B[I_{\text{qr.cs}}, I_{\text{qr.rs}}] = \begin{bmatrix} R \\ 0 \end{bmatrix}^T$
18: *% Perform trailing matrix updates*
19: $W = B[I_{\text{up.cs}}, I_{\text{qr.rs}}]UT, \quad V = -W$
20: $V[I_{\text{v.rs}}, :] = V[I_{\text{v.rs}}, :] + \frac{1}{2}U(T^T(U^T W[I_{\text{v.rs}}, :]))$
21: $B[I_{\text{qr.rs}}, I_{\text{up.cs}}] = B[I_{\text{qr.rs}}, I_{\text{up.cs}}] + UV^T$
22: $B[I_{\text{up.cs}}, I_{\text{qr.rs}}] = B[I_{\text{up.cs}}, I_{\text{qr.rs}}] + VU^T$

Ensure: B is a banded matrix with band-width h and the same eigenvalues as A

$k \leq 1 + p^{2-3\delta}$, *in BSP time,*

$$O\left(\gamma \cdot \frac{n^2 b}{p} + \beta \cdot \frac{n^{1+\delta} b^{1-\delta}}{p^\delta} + \alpha \cdot \frac{k^\delta n^{1-\delta} p^\delta}{b^{1-\delta}} \log p\right).$$

Proof. The cost of each inner loop iteration (loop on line 6) can be derived from the costs of the matrix multiplications and QR done inside it. Let the pair (i, j) correspond to the the ith iteration of the outer loop and jth iteration of the inner loop. Figure 2 displays the QR factorizations and updates computed during a few such iterations. Each iteration computes a QR factorization of a matrix with dimensions at most $(b-h) \times h$, $B[I_{\text{qr.rs}}, I_{\text{qr.cs}}]$ on line 16 with $\bar{p} = pb/(nk^{(1-\delta)/\delta})$ processors. The BSP time to compute such a QR factorization is by Theorem 3.6 for $\delta \in [1/2, 2/3]$,

$$O\left(\gamma \cdot \frac{bh^2}{\bar{p}} + \beta \cdot \frac{b^\delta h^{2-\delta}}{\bar{p}^\delta} + \alpha \cdot \bar{p}^\delta \log(\bar{p})\right)$$
$$= O\left(\gamma \cdot \frac{nb^2}{k^{3-1/\delta} p} + \beta \cdot \frac{n^\delta b^{2-\delta}}{kp^\delta} + \alpha \cdot k^{\delta-1}(pb/n)^\delta \log p\right)$$

The amount of memory needed for this QR factorization is given in Lemma 3.6 as

$$M = O\left((h^\delta b^{1-\delta}/\bar{p}^{1-\delta})^2\right) = O((n^{1-\delta} b^\delta/(p^{1-\delta} k^{(2\delta-1)/\delta}))^2).$$

The matrix multiplications to form the V matrix are on lines 19 and 20, while those to perform the updates are on lines 21 and 22. The matrix multiplications on line 20 should be done from right to left. We can then observe that the most costly matrix multiplications in Algorithm 4.2 are $B[I_{\text{up.cs}}, I_{\text{qr.rs}}]U$ on line 19 and the updates UV^T and VU^T on lines 21 and 22. In the first case, a $(3b-h) \times (b-h)$ is multiplied by a $(b-h) \times h$ matrix, while the update UV^T involve $(3b-h) \times h$ matrix multiplied by an $h \times (b-h)$ matrix (VU^T is just the transpose of the former). In both cases, by Lemma 3.2 with $v = \hat{p}^{2-3\delta}/(k-1)$ (we subtract one from k to make sure $v \geq 1$), the BSP time to compute the matrix multiplications using $\hat{p}$ processors is

$$O\left(\gamma \cdot \frac{b^2 h}{\hat{p}} + \beta \cdot \frac{b^2}{k\hat{p}^\delta} + \alpha \cdot \frac{\hat{p}^{2-3\delta}}{k} \log p\right)$$
$$= O\left(\gamma \cdot \frac{nb^2}{kp} + \beta \cdot \frac{n^\delta b^{2-\delta}}{kp^\delta} + \alpha \cdot \frac{(pb/n)^{2-3\delta}}{k} \log p\right),$$

with a memory footprint of $M = O(b^2/\hat{p} + (b^2 h/(v\hat{p}))^{2/3}) = O((b/\hat{p}^{1-\delta})^2) = O((n^{1-\delta} b^\delta/p^{1-\delta})^2)$, which is greater than the memory needed to perform the QR factorizations. The other matrix multiplications have strictly lower cost and the cost of redistributions necessary for all of these matrix multiplications is included in the horizontal communication cost of Lemma 3.2. As A and B are stored in load balanced layouts, each processor subset can obtain the submatrix which it factorizes and the submatrix which it updates at every iteration with $O(b^2/\hat{p})$ horizontal communication.

Thus, the overall cost for each iteration of Algorithm 4.2 is the sum of the two different costs above,

$$O\left(\gamma \cdot \frac{nb^2}{kp} + \beta \cdot \frac{n^\delta b^{2-\delta}}{kp} + \alpha \cdot k^{\delta-1}(pb/n)^\delta \log p\right).$$

For a given outer loop (line 4) iteration i, each j loop iteration (line 6) is done by a different processor group. The total number of inner loop iterations is roughly $(n/h)(n/b)/2$ and they are pipelined among n/b groups of processors, up to $n/(2b)$ of them working concurrently on different bulge chases at any given time. Consequently, the algorithm can be executed in $O(n/h)$ phases, where at the ith phase, $\min(i-1, (n-ih)/(2b))$ processor groups chase bulges concurrently and the ith panel is eliminated. At each phase, a synchronization and data exchange is required between the QR factorization and trailing matrix updates computed by adjacent active processor groups. Therefore, the BSP cost of each recursive step of the algorithm corresponds to the cost of computing $O(n/h) = O(kn/b)$ inner loop iterations using one processor group, which corresponds to the cost postulated in the lemma. □

4.3 Complete Symmetric Eigensolver

Algorithm 4.3 combines our algorithms for full-to-band reduction (Algorithm 4.1) with multiple subsequent stages of band-to-band reduction (Algorithm 4.2) and band-halving steps of the CA-SBR algorithm from [8], which we refer to as CA-BR. Algorithm 4.1 reduces the symmetric matrix to one with band-width at most

Algorithm 4.3 $[D] \leftarrow$ 2.5D-Symmetric-Eigensolver(A, Π)

Require: Given positive integers p, n, and $\delta \in [1/2, 2/3]$ with $n \bmod b \equiv 0$, A is a symmetric matrix of dimension n.

1: Let $b = \frac{n}{\max(p^{2-3\delta}, \log p)}$, $k = 2$, and $\zeta = (1-\delta)/\delta$
2: Execute $B =$ 2.5D-Full-to-Band$(A, \{\}, \{\}, \Pi, b)$
3: **for** $i = 0 : \log_2(bp^\delta/n) - 1$ **do**
4: Let $\bar{\Pi} = \Pi[1 : p/k^{i\zeta}]$
5: Gather B onto $\bar{\Pi}$
6: Execute $B =$ 2.5D-Band-to-Band$(B, b/k^i, \bar{\Pi}, k)$
7: Let $\bar{\Pi} = \Pi[1 : p^\delta]$
8: **for** $i = 0 : \log_2(p^{1-\delta}) - 1$ **do**
9: Execute $B =$ CA-BR$(B, n/(p^\delta k^i), \bar{\Pi}, k)$
10: Gather B onto a processor and compute its eigenvalues D

Ensure: D is a vector containing the eigenvalues of A

$n/\log p$. Algorithm 4.2 is then used to successively half the band-width to n/p^δ. Subsequently, the CA-BR algorithm (same function signature as 2.5D-Band-to-Band) is used to reduce the band-width to n/p. At that point, the matrix is small enough for one processor to compute the eigenvalues efficiently.

For every 2.5D-Band-to-Band step that reduces the band-width by a factor of k, Algorithm 4.2 reduces the number of processors used by k^ζ where $\zeta = (1-\delta)/\delta$. The parameter ζ is chosen to be $(1-\delta)/\delta$ so that the per-stage horizontal cost term $O(nb/p^\delta)$ does not increase at each recursive step, since $n(b/k)/(p/k^\zeta)^\delta = nb/p^\delta$. Decreasing the number of active processors in this way also keeps the synchronization cost equal at every stage. Overall, we now obtain a parallel algorithm that has horizontal communication of $O(n^2/p^\delta)$, vertical communication of $O(n^2 \log p/p^\delta)$, and $O(p^\delta \log^2 p)$ synchronizations. Modulo logarithmic cost factors in vertical communication and synchronization, this amounts to the same communication cost as the best known algorithms for LU and QR factorization [3, 35, 38].

Theorem 4.4. *Algorithm 4.3 computes the eigenvalues of a symmetric n-by-n matrix (input in any evenly-distributed layout), using $M = O(n^2/p^{2(1-\delta)})$ memory for any $\delta \in [1/2, 2/3]$, in BSP time,*

$$O\left(\gamma \cdot \frac{n^3}{p} + \beta \cdot \frac{n^2}{p^\delta} + \nu \cdot \frac{n^2 \log p}{p^\delta} + \alpha \cdot p^\delta \log^2 p\right).$$

Proof. The cost of the gather/redistribution of B onto $\bar{\Pi}$ is dominated by the subsequent 2.5D-Band-to-Band invocation. The cost of computing the eigenvalues of B sequentially at the end is $O(\gamma \cdot n^3/p + \beta \cdot n^2/p + \alpha)$, since the band-width is n/p [8]. We employ Lemma 4.1 with $b = \frac{n}{\max(p^{2-3\delta}, \log p)}$ to obtain the cost of 2.5D-Full-to-Band. The computation, horizontal communication, and synchronization costs are the same for 2.5D-Full-to-Band as the ones postulated in Theorem 4.4. The vertical communication cost term incurred for small cache sizes, $O(\nu \cdot (n/b)n^2/p^{2(1-\delta)})$ is bounded by $O(\nu \cdot [n^2/p^\delta + n^2 \log p/p^{2/3}]) = O(\nu \cdot n^2 \log p/p^\delta)$.

We now consider the memory footprint and cost of the invocations of 2.5D-Band-to-Band. By Lemma 4.3 with $k = 2$, the memory usage is $M = O((n^{1-\delta}\bar{b}^\delta/\bar{p}^{1-\delta})^2)$, where $\bar{b} = b/k^i$ where $\bar{p} = p/k^{i\zeta}$ at iteration i. We observe that $(n^{1-\delta}\bar{b}^\delta/\bar{p}^{1-\delta})^2 = O(n^2/p^{2(1-\delta)})$ for all iterations i, because at each subsequent iteration $\bar{b}$ decreases by k while $\bar{p}$ decreases by k^ζ, and so $\bar{b}^\delta/\bar{p}^{1-\delta} \le b^\delta/p^{1-\delta} \le n^\delta/p^{1-\delta}$ for all i, since $k^{(1-\delta)\zeta}/k^\delta = k^{(1-\delta)^2/\delta}/k^\delta \le k^{1-\delta}/k^\delta \le 1$. The cost of each band reduction with starting band-width $\bar{b}$ and $\bar{p}$ processors is by Lemma 4.3 with $k = 2$,

$$O\left(\gamma \cdot \frac{n^2\bar{b}}{\bar{p}} + \beta \cdot \frac{n^{1+\delta}\bar{b}^{1-\delta}}{\bar{p}^\delta} + \alpha \cdot \frac{n^{1-\delta}\bar{p}^\delta}{\bar{b}^{1-\delta}} \log p\right).$$

The computation cost clearly decreases with each iteration i. The horizontal communication cost is $O(nb/p^\delta) = O(n^2/(p^\delta \log p))$ (since $b \le n/\log p$) at each iteration, since

$$\frac{\bar{b}^{1-\delta}}{\bar{p}^\delta} = \frac{(b/k^i)^{1-\delta}}{(p/k^{i\zeta})^\delta} = \frac{b^{1-\delta}}{p^\delta}.$$

Therefore, over all $O(\log p)$ iterations, the bandwidth cost of the SBR invocations is $O(n^2/p^\delta)$. Finally, the synchronization cost is $\frac{n^{1-\delta}\bar{p}^\delta}{\bar{b}^{1-\delta}} \log p = O(p^\delta \log p)$ at each iteration, since $\bar{p}^\delta/\bar{b}^{1-\delta} = p^\delta/b^{1-\delta}$. Thus, the overall synchronization cost is as postulated.

The time for CA-BR using p^δ processors starting from band-width n/p^δ and reducing it to n/p is via Lemma 4.2, $O(\gamma \cdot \frac{n^3}{p^{2\delta}} + \beta \cdot \frac{n^2}{p^\delta} + \nu \cdot \frac{n^2 \log p}{p^\delta} + \alpha \cdot p^\delta \log p)$. Computing the eigenvalues of a matrix with band-width n/p sequentially costs $O(\gamma \cdot \frac{n^3}{p})$ [8]. □

Algorithm	W (β)	Q (ν)	S (α)
ScaLAPACK [12]	$n^2/\sqrt{p}$	n^3/p	$n \log p$
ELPA [4]	$n^2/\sqrt{p}$	-	$n \log p$
CA-SBR [8]	$n^2/\sqrt{p}$	$n^2 \log n/\sqrt{p}$	$\sqrt{p}(\log^2 p + \log n)$
Theorem 4.4	n^2/p^δ	$n^2 \log p/p^\delta$	$p^\delta \log^2 p$

Table 1: Asymptotic costs for computing eigenvalues, with $\delta \in [1/2, 2/3]$. All variants have $O(n^3/p)$ computation cost.

A disadvantage of this multi-stage approach arises when eigenvectors are required in addition to eigenvalues. The cost of the back-transformations scales linearly with the number of band-reduction stages (each stage requires $O(n^2)$ memory and $O(n^3)$ computation). We leave the consideration of eigenvector construction for future work. To reduce the number of band-reduction stages when $\delta < 2/3$, one can use $k = p^{2-3\delta}$ with each invocation of 2.5D-Band-to-Band, but this results in a greater synchronization cost. It may also be possible to improve the 2.5D-Band-to-Band algorithm by using aggregation as in the 2.5D-Full-to-Band algorithm.

5 CONCLUSION

Table 1 provides a comparison of communication and synchronization costs to previous work. Our new direct method for computing the eigenvalues of a symmetric matrix, performs up to $p^{1/6}$ less horizontal communication than alternatives. The vertical communication cost (Q) for ScaLAPACK assumes $H < n^2/p$ and arises from the matrix-vector multiplications computing V for each column. For CA-SBR, Q is inferred from Lemma 4.2. For ELPA, we assume the full-to-band step reduces to band-width $b = \sqrt{H}$, in which case either (when $\sqrt{H} > n/p$) the banded matrix fits in cache, or $\nu \cdot Q = O(\nu \cdot [n^3/(pb) + nb^2]) = O(\gamma \cdot F/\sqrt{H})$ [4].

The new 2.5D-Symmetric-Eigensolver algorithm trades off a variable amount of extra work, synchronization, and memory usage for a lower communication cost. Implementations of the algorithms in this paper permit optimizations such as

- alternating between left-looking partial updates and complete trailing matrix updates in Algorithm 4.1,
- smaller bulge width in Algorithm 4.2 to increase parallelism in the bulge chase pipeline,
- lookahead [2, 36] (overlapping QR with updates).

Our analysis shows that a carefully parameterized collage of parallel algorithms and optimizations yields asymptotic cost improvements with minimal overhead. We combine approaches (2.5D algorithms, aggregation, successive band reduction) that have been successful on modern architectures [5, 6, 33], so our innovations should pave the path for practical improvements in scalability of applications computing singular values or eigenvalues of matrices.

REFERENCES

[1] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. 1995. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development* 39 (September 1995), 575–582. Issue 5.

[2] Ramesh C Agarwal and Fred G Gustavson. 1988. A parallel implementation of matrix multiplication and LU factorization on the IBM 3090. In *Proceedings of the IFIP WG*, Vol. 2. 217–221.

[3] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. 1990. Communication complexity of PRAMs. *Theoretical Computer Science* 71, 1 (1990), 3 – 28.

[4] Thomas Auckenthaler. 2012. *Highly scalable eigensolvers for petaflop applications*. Ph.D. Dissertation. Universität München.

[5] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, and P. Willems. 2011. Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem. *Journal of Computational Science* 2, 3 (2011), 272 – 278. https://doi.org/10.1016/j.jocs.2011.05.002 Social Computational Systems.

[6] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H. D. Nguyen, and E. Solomonik. 2014. Reconstructing Householder Vectors from Tall-Skinny QR. In *Proceedings of the 28th IEEE International Symposium on Parallel and Distributed Processing (IPDPS '14)*. 1159–1170. https://doi.org/10.1109/IPDPS.2014.120

[7] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2011. Minimizing Communication in Numerical Linear Algebra. *SIAM J. Matrix Anal. Appl.* 32, 3 (2011), 866–901. https://doi.org/10.1137/090769156

[8] Grey Ballard, James Demmel, and Nicholas Knight. 2015. Avoiding Communication in Successive Band Reduction. *ACM Transactions on Parallel Computing* 1, 2, Article 11 (Feb. 2015), 37 pages. https://doi.org/10.1145/2686877

[9] Jarle Berntsen. 1989. Communication efficient matrix multiplication on hypercubes. *Parallel Comput.* 12, 3 (1989), 335–342.

[10] C. Bischof, B. Lang, and X. Sun. 2000. Algorithm 807: The SBR Toolbox – Software Successive Band Reduction. *ACM Trans. Math. Software* 26, 4 (Dec 2000), 602–616.

[11] C. Bischof, B. Lang, and X. Sun. 2000. A Framework for Symmetric Band Reduction. *ACM Trans. Math. Software* 26, 4 (Dec 2000), 581–601.

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

[13] Rudnei Dias da Cunha, Dulcenéia Becker, and James Carlton Patterson. 2002. New parallel (rank-revealing) QR factorization algorithms. In *Euro-Par 2002 Parallel Processing*. Springer, 677–686.

[14] Eliezer Dekel, David Nassimi, and Sartaj Sahni. 1981. Parallel Matrix and Graph Algorithms. *SIAM J. Comput.* 10, 4 (1981), 657–675.

[15] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger. 2013. Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication. In *Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing (IPDPS '13)*. 261–272. https://doi.org/10.1109/IPDPS.2013.80

[16] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. 2012. Communication-optimal Parallel and Sequential QR and LU Factorizations. *SIAM Journal on Scientific Computing* 34, 1 (2012), A206–A239. https://doi.org/10.1137/080731992

[17] Inderjit S. Dhillon, Beresford N. Parlett, and Christof Vömel. 2006. The Design and Implementation of the MRRR Algorithm. *ACM Trans. Math. Software* 32, 4 (Dec. 2006), 533–560. https://doi.org/10.1145/1186785.1186788

[18] Jack J Dongarra, Danny C Sorensen, and Sven J Hammarling. 1989. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math.* 27, 1 (1989), 215–227.

[19] Jack J Dongarra and Robert A van de Geijn. 1992. Reduction to condensed form for the Eigenvalue problem on distributed memory architectures. *Parallel Comput.* 18, 9 (1992), 973 – 982. https://doi.org/10.1016/0167-8191(92)90011-U

[20] E. Elmroth and F. Gustavson. 1998. New serial and parallel recursive QR factorization algorithms for SMP systems. In *Applied Parallel Computing. Large Scale Scientific and Industrial Problems.*, B. Kågström et al. (Ed.). Lecture Notes in Computer Science, Vol. 1541. Springer, 120–128.

[21] V. Fock. 1930. Näherungsmethode zur Lösung des quantenmechanischen Mehrkörperproblems. *Zeitschrift für Physik* 61, 1-2 (1930), 126–148. https://doi.org/10.1007/BF01340294

[22] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. 1999. Cache-Oblivious Algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*. IEEE Computer Society, Washington, DC, USA, 285.

[23] Gene H Golub, Robert J Plemmons, and Ahmed Sameh. 1986. *Parallel block schemes for large-scale least-squares computations*. University of Illinois Press. 171–179 pages.

[24] Brian C. Gunter and Robert A. Van De Geijn. 2005. Parallel Out-of-core Computation and Updating of the QR Factorization. *ACM Trans. Math. Software* 31, 1 (March 2005), 60–78. https://doi.org/10.1145/1055531.1055534

[25] Azzam Haidar, Hatem Ltaief, and Jack Dongarra. 2011. Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems Using Aggregated Fine-grained and Memory-aware Kernels. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. ACM, New York, NY, USA, Article 8, 11 pages. https://doi.org/10.1145/2063384.2063394

[26] D. R. Hartree. 1928. The Wave Mechanics of an Atom with a Non-Coulomb Central Field. Part I. Theory and Methods. *Mathematical Proceedings of the Cambridge Philosophical Society* 24 (1 1928), 89–110. Issue 01. https://doi.org/10.1017/S0305004100011919

[27] Hong Jia-Wei and H. T. Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing (STOC '81)*. ACM, New York, NY, USA, 326–333.

[28] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Ortí, Robert van de Geijn, and Field G. Van Zee. 2006. Accumulating Householder Transformations, Revisited. *ACM Trans. Math. Software* 32, 2 (June 2006), 169–179. https://doi.org/10.1145/1141885.1141886

[29] S. Lennart Johnsson. 1993. Minimizing the communication time for matrix multiplication on multiprocessors. *Parallel Comput.* 19 (November 1993), 1235–1257. Issue 11.

[30] B. Lang. 1993. A Parallel Algorithm for Reducing Symmetric Banded Matrices to Tridiagonal Form. *SIAM Journal on Scientific Computing* 14, 6 (1993), 1320–1338. https://doi.org/10.1137/0914078

[31] W. F. McColl and A. Tiskin. 1999. Memory-Efficient Matrix Multiplication in the BSP Model. *Algorithmica* 24 (1999), 287–297. Issue 3.

[32] Edgar Solomonik. 2014. *Provably Efficient Algorithms for Numerical Tensor Algebra*. Ph.D. Dissertation. University of California, Berkeley.

[33] Edgar Solomonik, Abhinav Bhatele, and James Demmel. 2011. Improving Communication Performance in Dense Linear Algebra via Topology Aware Collectives. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. ACM, New York, NY, USA, Article 77, 11 pages. https://doi.org/10.1145/2063384.2063487

[34] Edgar Solomonik, Erin Carson, Nicholas Knight, and James Demmel. 2014. Tradeoffs Between Synchronization, Communication, and Computation in Parallel Linear Algebra Computations. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 307–318. https://doi.org/10.1145/2612669.2612671

[35] Edgar Solomonik and James Demmel. 2011. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. In *Euro-Par 2011 Parallel Processing*. Lecture Notes in Computer Science, Vol. 6853. Springer Berlin Heidelberg, 90–109. https://doi.org/10.1007/978-3-642-23397-5_10

[36] Peter Strazdins. 2001. A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization. *International Journal Parallel and Distributed Systems and Networks* 4, 1 (2001), 26–35.

[37] A. Tiskin. 2002. Bulk-Synchronous Parallel Gaussian Elimination. *Journal of Mathematical Sciences* 108 (2002), 977–991. Issue 6. https://doi.org/10.1023/A:1013588221172

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

[39] Leslie G Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (1990), 103–111.

[40] R. A. Van De Geijn and J. Watts. 1997. SUMMA: Scalable Universal Matrix Multiplication Algorithm. *Concurrency: Practice and Experience* 9, 4 (1997), 255–274.