Algorithms for Reducing a Matrix to Condensed Form

FLAME Working Note #53

Field G. Van Zee*
Robert A. van de Geijn*
Gregorio Quintana-Ortí†
G. Joseph Elizondo*

October 30, 2010

Abstract

In a recent paper it was shown how memory traffic can be diminished by reformulating the classic algorithm for reducing a matrix to bidiagonal form, a preprocess when computing the singular values of a dense matrix. The key is a reordering of the computation so that the most compute- and memory-intensive operations can be "fused". In this paper, we show that other operations that reduce matrices to condensed form (reduction to upper Hessenberg and reduction to ridiagonal form) can be similarly reorganized, yielding different sets of operations that can be fused. By developing the algorithms with a common framework and notation, we facilitate the comparing and contrasting of the different algorithms and opportunities for optimization. We discuss the algorithms and showcase the performance improvements that they facilitate.

1 Introduction

For many dense linear algebra operations there exist algorithms that cast most computation in term of matrix-matrix operations that overcome the memory bandwidth bottleneck in current processors [9, 8, 6, 1]. Reduction to condensed form operations are important exceptions. For these operations reducing the number of times data must be brought in from memory is the key to optimizing performance since inherently $O(n^3)$ reads and writes from memory are incurred while $O(n^3)$ floating-point operations are performed on an $n \times n$ matrix.

The Basic Linear Algebra Subprograms (BLAS) [15, 7, 6] provide an interface to commonly used computational kernels in terms of which linear algebra routine can be written. The idea is that if these kernels are optimized, then implementations of algorithms for computing more complex operations benefit in a portable fashion. As we will see, the problem is that the interface itself is limiting and can stand in the way of minimizing memory traffic. In response, as part of the BLAST Forum [5], additional, more complex, operations were suggested for inclusion in the BLAS. Unfortunately, the extensions proposed by the BLAST forum are not as well-supported as the original BLAS. In [12], it was shown how one of the reduction to condensed form operations, reduction to bidiagonal form, benefits from this new functionality in the BLAS.

The present paper presents algorithms for all three major reduction to condensed form operations (reduction to upper Hessenberg, tridiagonal, and bidiagonal form) with the FLAME notation [10]. This facilitates comparing and contrasting of different algorithms for the same operation and similar algorithms for different operations [18, 10, 2, 23]. It shows how the techniques used to reduce memory traffic in the reduction to bidiagonal form algorithm, already reported in [12], can be modified to similarly reduce such traffic when computing a reduction to upper Hessenberg or tridiagonal form, although with less practical success. It identifies sets of operations that can be fused in an effort to reduce the cost due to memory traffic of the three

*Department of Computer Science, The University of Texas at Austin, Austin, TX, 78712.
†Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Campus Riu Sec, 12.071, Castellón, Spain,

algorithms for reduction to condensed form. Such operations have been referred to as "Level-2.5 BLAS". It demonstrates the relative merits of different algorithms and optimizations that combine algorithms. All the presented algorithms are implemented as part of the `libflame` library [24, 25]. Thus the paper provides documentation for that library's support of the target operations. The family of implementations and related benchmarking codes are available as part of `libflame` so that others can experiment with optimizations of the fused operations and the effect on performance.

This paper is structured as follows: In Section 2 we discuss the Householder transform, including some of its properties we will use later in the paper. Various algorithms for reducing a matrix to upper Hessenberg form are developed in Section 3, including a discussion on how to fuse key matrix-vector operations to reduce memory traffic. Section 4 briefly discusses reduction to tridiagonal form and how it is similar to its upper Hessenberg counterpart. The third operation, reduction to bidiagonal form, is discussed in Section 5. Performance is discussed in Section 6 and concluding remarks can be found in Section 7. In Appendix A, we introduce a complex Householder transform and give examples of how generalizing to the complex domain affects the various reduction algorithms.

2 Householder transformations (Reflectors)

We start by reviewing a few basic properties of Householder transformations.

2.1 Computing Householder vectors and transformations

Definition 1 *Let $u \in \mathbb{R}^n$, $\tau \in \mathbb{R}$. Then $H = H(u) = I - uu^T/\tau$, where $\tau = \frac{1}{2}u^T u$, is said to be a reflector or Householder transformation.*

We observe:

- Let z be any vector that is perpendicular to u. Applying a Householder transform $H(u)$ to z leaves the vector unchanged: $H(u)z = z$.
- Let any vector x be written as $x = z + u^T xu$, where z is perpendicular to u and $u^T xu$ is the component of x in the direction of u. Then $H(u)x = z - u^T xu$.

This can be interpreted as follows: The space perpendicular to u acts as a "mirror": any vector in that space (along the mirror) is not reflected, while any other vector has the component that is orthogonal to the space (the component outside and orthogonal to the mirror) reversed in direction. Notice that a reflection preserves the length of the vector. Also, it is easy to verify that:

1. $HH = I$ (reflecting the reflection of a vector results in the original vector);
2. $H = H^T$, and so $H^T H = HH^T = I$ (a reflection is an orthogonal matrix and thus preserves the norm); and
3. if $H_0, \cdots, H_{k-1}$ are Householder transformations and $Q = H_0 H_1 \cdots H_{k-1}$, then $Q^T Q = QQ^T = I$ (an accumulation of reflectors is an orthogonal matrix).

As part of the reduction to condensed form operations, given a vector x we will wish to find a Householder transformation, $H(u)$, such that $H(u)x$ equals a vector with zeroes below the first element: $H(u)x = \mp\|x\|_2 e_0$ where e_0 equals the first column of the identity matrix. It can be easily checked that choosing $u = x \pm \|x\|_2 e_0$ yields the desired $H(u)$. Notice that any nonzero scaling of u has the same property, and the convention is to scale u so that the first element equals one. Let us define $[u, \tau, h] = \textsc{Housev}(x)$ to be the function that returns u with first element equal to one, $\tau = \frac{1}{2}u^T u$, and $h = H(u)x$.

2.2 Computing Au from Ax

Later, we will see that given a matrix A, we will need to form Au where u is computed by $\textsc{Housev}(x)$, but we will do so by first computing Ax. Let

$$x \rightarrow \left(\begin{array}{c} \chi_1 \\ \hline x_2 \end{array} \right), \quad v \rightarrow \left(\begin{array}{c} \nu_1 \\ \hline v_2 \end{array} \right), \quad u \rightarrow \left(\begin{array}{c} \upsilon_1 \\ \hline u_2 \end{array} \right),$$

$v = x - \alpha e_0$ and $u = v/\nu_1$, with $\alpha = -\mathrm{sign}(\chi_1)\|x\|_2$. Then

$$\begin{aligned} \|x\|_2 &= \left\| \left(\begin{array}{c} \chi_1 \\ \hline \|x_2\|_2 \end{array} \right) \right\|_2, \quad \|v\|_2 = \left\| \left(\begin{array}{c} \chi_1 - \alpha \\ \hline \|x_2\|_2 \end{array} \right) \right\|_2, \quad \|u\|_2 = \left\| \left(\begin{array}{c} \|v\|_2 \\ \hline \chi_1 - \alpha \end{array} \right) \right\|_2, & (1) \\ \tau &= \frac{u^T u}{2} = \frac{\|u\|_2^2}{2} = \frac{\|v\|_2^2}{2(\chi_1 - \alpha)^2}, & (2) \\ w &= Ax \quad \text{and} \quad Au = \frac{A(x - \alpha e_0)}{(\chi_1 - \alpha)} = \frac{(w - \alpha A e_0)}{(\chi_1 - \alpha)}. & (3) \end{aligned}$$

We note that Ae_0 simply equals the first column of A. We will assume that various results in Eq. (1)–(2) are computed by the function $\textsc{Houses}(x)$ where $[\chi_1 - \alpha, \tau, \alpha] = \textsc{Houses}(x)$.[1] Then, the desired vector Au can be computed via Eq. (3).

2.3 Accumulating transformations

Consider the transformation formed by multiplying b Householder transformations $\left(I - u_j u_j^T/\tau_j\right)$, for $0 \leq j < b-1$. In [13] it was shown that if $U = \left(\begin{array}{c|c|c|c} u_0 & u_1 & \cdots & u_{b-1} \end{array} \right)$, then

$$\left(I - u_0 u_0^T/\tau_0\right)\left(I - u_1 u_1^T/\tau_1\right) \cdots \left(I - u_{b-1} u_{b-1}^T/\tau_{b-1}\right) = (I - UT^{-1}U^T).$$

Here $T = \frac{1}{2}D + S$ where D and S equal the diagonal and strictly upper triangular parts of $U^TU = S^T + D + S$. Later we will use the fact that if

$$U = \left(\begin{array}{c|c} U_0 & u_1 \end{array} \right) \quad \text{and} \quad T = \left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_{11} \end{array} \right)$$

then

$$t_{01} = U_0^T u_1, \quad \tau_{11} = \frac{u_{21}^T u_{21}}{2}, \quad \text{and} \quad \left(\begin{array}{c|c} T_{00} & t_{01} \\ \hline 0 & \tau_{11} \end{array} \right)^{-1} = \left(\begin{array}{c|c} T_{00}^{-1} & -T_{00}^{-1} t_{01}/\tau_{11} \\ \hline 0 & \tau_{11}^{-1} \end{array} \right).$$

For further details, see [13, 17, 22, 27]. Alternative ways for accumulating transformations are the WY-transform [4] and compact WY-transform [20].

3 Reduction to upper Hessenberg form

In the first step towards computing the Schur decomposition of a matrix A, the matrix is reduced to upper Hessenberg form: $A \rightarrow QBQ^T$ where B is an upper Hessenberg matrix (zeroes below the first subdiagonal) and Q is orthogonal.

3.1 Unblocked algorithm

The basic algorithm for reducing the matrix to upper Hessenberg form, overwriting the original matrix with the result, can be explained as follows.

[1] Here, $\textsc{Houses}$ stands for "Householder scalars", in contrast to the function $\textsc{Housev}$ which provides the Householder vector u.

Algorithm: $[A] := \text{HessRed\_unb}(b, A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $u \rightarrow \left(\begin{array}{c} u_T \\ \hline u_B \end{array}\right)$, $y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$, $z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$

where A_{TL} is 0×0 and u_T, y_T, and z_T have 0 rows

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} u_T \\ \hline u_B \end{array}\right) \rightarrow \left(\begin{array}{c} u_{01} \\ \hline \upsilon_{11} \\ \hline u_{21} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_{01} \\ \hline \psi_{11} \\ \hline y_{21} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_{01} \\ \hline \zeta_{11} \\ \hline z_{21} \end{array}\right)$$

where α_{11}, υ_{11}, ψ_{11}, ζ_{11} are scalars

Basic unblocked 1:

$[u_{21}, \tau, a_{21}] := \text{Housev}(a_{21})$

$A_{22} := (I - u_{21}u_{21}^T/\tau)A_{22} = A_{22} - u_{21}u_{21}^T A_{22}/\tau$

$$\left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right)(I - u_{21}u_{21}^T/\tau) = \left(\begin{array}{c} A_{02} - A_{02}u_{21}u_{21}^T/\tau \\ \hline a_{12}^T - a_{12}^T u_{21}u_{21}^T/\tau \\ \hline A_{22} - A_{22}u_{21}u_{21}^T/\tau \end{array}\right)$$

Basic unblocked 2:

$[u_{21}, \tau, a_{21}] := \text{Housev}(a_{21})$

$y_{21} := A_{22}^T u_{21}$
$z_{21} := A_{22}u_{21}$

$\beta := u_{21}^T z_{21}/2$
$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$
$z_{21} := (z_{21} - \beta u_{21}/\tau)/\tau$
$A_{22} := A_{22} - u_{21}y_{21}^T - z_{21}u_{21}^T$
$a_{12}^T := a_{12}^T - a_{12}^T u_{21}u_{21}^T/\tau$
$A_{02} := A_{02} - A_{02}u_{21}u_{21}^T/\tau$

Rearranged unblocked:

$\alpha_{11} := \alpha_{11} - \upsilon_1\psi_1 - \zeta_1\upsilon_1$ $(\star)$
$a_{12}^T := a_{12}^T - \upsilon_1 y_{21}^T - \zeta_1 u_{21}^T$ $(\star)$
$a_{21} := a_{21} - u_{21}\psi_1 - z_{21}\upsilon_1$ $(\star)$
$[x_{21}, \tau, a_{21}] := \text{Housev}(a_{21})$
$A_{22} := A_{22} - u_{21}y_{21}^T - z_{21}u_{21}^T$ $(\star)$
$v_{21} := A_{22}^T x_{21}$
$w_{21} := A_{22}x_{21}$
$u_{21} := x_{21}$; $y_{21} := v_{21}$
$z_{21} := w_{21}$
$\beta := u_{21}^T z_{21}/2$
$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$
$z_{21} := (z_{21} - \beta u_{21}/\tau)/\tau$

$a_{12}^T := a_{12}^T - a_{12}^T u_{21}u_{21}^T/\tau$
$A_{02} := A_{02} - A_{02}u_{21}u_{21}^T/\tau$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} u_T \\ \hline u_B \end{array}\right) \leftarrow \left(\begin{array}{c} u_{01} \\ \hline \upsilon_{11} \\ \hline u_{21} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_{01} \\ \hline \psi_{11} \\ \hline y_{21} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_{01} \\ \hline \zeta_{11} \\ \hline z_{21} \end{array}\right)$$

endwhile

Figure 1: Unblocked algorithms for reduction to upper Hessenberg form. Operations marked with $(\star)$ are not executed during the first iteration.

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right)$.
- Let $[u_{21}, \tau, a_{21}] := \text{HOUSEV}(a_{21})$.[2]
- Update

$$\left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & H \end{array}\right) \left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & H \end{array}\right) = \left(\begin{array}{c|c} a_{01} & A_{02}H \\ \hline \alpha_{11} & a_{12}^T H \\ \hline Ha_{21} & HA_{22}H \end{array}\right)$$

  where $H = H(u_{21})$. Note that $a_{21} := Ha_{21}$ need not be executed since this update was performed by the instance of HOUSEV above.[3]
- Continue this process with the updated A_{22}.

This is captured in the algorithm in Figure 1 (top), in which it is recognized that as the algorithm proceeds beyond the first iteration, the submatrix A_{20} must also be updated. As formulated, the submatrix A_{22} has to be read and written in the first highlighted operation and submatrices A_{02}, a_{12}^T, and A_{22} must be read and written in the second highlighted operation in Figure 1 (top) *if the operations in the highlighted boxed are "fused" by which we mean that they are implemented at the same level as a typical level-2 BLAS operation.* Thus, the bulk of memory operations then lie with A_{22} being read and written twice and A_{20} being read and written once. We will track the number of times A_{22} and A_{20} need to be read and written by the different algorithms in Figure 2.

Let us look at the update of A_{22} in Figure 1 (top) in more detail:

$$\begin{aligned} A_{22} &:= HA_{22}H = (I - u_{21}u_{21}^T/\tau)A_{22}(I - u_{21}u_{21}^T/\tau) \\ &= A_{22} - u_{21}(\underbrace{A_{22}^T u_{21}}_{v_{21}})^T/\tau - (\underbrace{A_{22}u_{21}}_{w_{21}})u_{21}^T/\tau + (u_{21}^T \underbrace{A_{22}u_{21}}_{w_{21}})u_{21}u_{21}^T/\tau^2 \\ &= A_{22} - u_{21}v_{21}^T/\tau - w_{21}u_{21}^T/\tau + \underbrace{u_{21}^T w_{21}}_{2\beta} u_{21}u_{21}^T/\tau^2 \\ &= A_{22} - u_{21}\underbrace{((v_{21} - \beta u_{21}/\tau)/\tau))^T}_{y_{21}} - \underbrace{((w_{21} - \beta u_{21}/\tau)/\tau)}_{z_{21}} u_{21}^T \\ &= A_{22} - (u_{21}y_{21}^T + z_{21}u_{21}^T). \end{aligned}$$

This motivates the algorithm in Figure 1 (left). The problem with this algorithm is that, when implemented using traditional level-2 BLAS, it requires A_{22} to be read four times and written twice. If the operations in the highlighted boxes are instead fused, then A_{22} needs only be read twice and written once.

What we will show next is that by delaying the update $A_{22} := A_{22} - (u_{21}y_{21}^T + z_{21}u_{21}^T)$ until the next iteration, we can reformulate the algorithm so that A_{22} needs only be read and written once per iteration. Let us focus on the update $A_{22} := A_{22} - (u_{21}y_{21}^T + z_{21}u_{21}^T)$. Partition

$$A_{22} \rightarrow \left(\begin{array}{c|c} \alpha_{11}^+ & a_{12}^{+T} \\ \hline a_{21}^+ & A_{22}^+ \end{array}\right), \quad u_{21} \rightarrow \left(\begin{array}{c} \upsilon_1^+ \\ \hline u_{21}^+ \end{array}\right), \quad y_{21} \rightarrow \left(\begin{array}{c} \psi_1^+ \\ \hline y_{21}^+ \end{array}\right), \quad z_{21} \rightarrow \left(\begin{array}{c} \zeta_1^+ \\ \hline z_{21}^+ \end{array}\right),$$

where $+$ indicates the partitioning in the next iteration. Then $A_{22} := A_{22} - (u_{21}y_{21}^T + z_{21}u_{21}^T)$ translates to

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11}^+ & a_{12}^{+T} \\ \hline a_{21}^+ & A_{22}^+ \end{array}\right) &:= \left(\begin{array}{c|c} \alpha_{11}^+ & a_{12}^{+T} \\ \hline a_{21}^+ & A_{22}^+ \end{array}\right) - \left(\left(\begin{array}{c} \upsilon_1^+ \\ \hline u_{21}^+ \end{array}\right) \left(\begin{array}{c} \psi_1^+ \\ \hline y_{21}^+ \end{array}\right)^T + \left(\begin{array}{c} \zeta_1^+ \\ \hline z_{21}^+ \end{array}\right) \left(\begin{array}{c} \upsilon_1^+ \\ \hline u_{21}^+ \end{array}\right)^T \right) \\ &= \left(\begin{array}{c|c} \alpha_{11}^+ - (\upsilon_1^+\psi_1^+ + \zeta_1^+\upsilon_1^+) & a_{12}^{+T} - (\upsilon_1^+ y_{21}^{+T} + \zeta_1^+ u_{21}^{+T}) \\ \hline a_{21}^+ - (u_{21}^+\psi_1^+ + z_{21}^+\upsilon_1^+) & A_{22}^+ - (u_{21}^+ y_{21}^{+T} + z_{21}^+ u_{21}^{+T}) \end{array}\right), \end{aligned}$$

[2]Note that the semantics here indicate that a_{21} is overwritten by Ha_{21}.

[3]In practice, the zeros below the first element of Ha_{21} are not actually written. Instead, the implementation overwrites these elements with the corresponding elements of the vector u_{21}.

Algorithm		Read		Write	
		A_{22}	A_{02}	A_{22}	A_{02}
Reduction to Hessenberg form					
Basic unblocked 1	Unfused ($\star$)	4	2	2	1
	Fused	2	1	2	1
Basic unblocked 2	Unfused ($\star$)	4	2	2	1
	Fused	2	1	1	1
Rearranged unblocked	Unfused ($\star$)	4	2	2	1
	Fused	1	1	1	1
Blocked + basic unblocked 2	Unfused ($\star$)	4	$2/b$	2	$1/b$
	Fused	2	$2/b$	1	$1/b$
Blocked + rearranged unblocked	Unfused ($\star$)	4	$2/b$	2	$1/b$
	Fused	1	$2/b$	1	$1/b$
Blocked + lazy unblocked	Unfused ($\star$)	$2 + 2/b$	$2/b$	$2/b$	$1/b$
	Fused	$1 + 2/b$	$2/b$	$2/b$	$1/b$
GQvdG blocked + GQvdG unblocked ($\star$)		$1 + 3/b$	$2/b$	$2/b$	$1/b$
Reduction to tridiagonal form					
Basic unblocked		2		1	
Rearranged unblocked	Unfused ($\star$)	2		1	
	Fused	1		1	
Blocked + lazy unblocked ($\star$)		$1 + 1/b$		$1/b$	
Reduction to bidiagonal form					
Basic unblocked	Unfused ($\star$)	4		2	
	Fused	2		1	
Rearranged unblocked	Unfused ($\star$)	4		2	
	Fused	1		1	
Howell's Algorithm		$1 + 2/b$		$2/b$	

Figure 2: Summary of the number of times the different major submatrices of A must be brought in from memory per column of A. (The ($\star$) indicates that the indicated algorithm does NOT require fused operations. In other words, traditional level-2 BLAS suffice). There are opportunities for fusing level-3 BLAS as well, which is not explored in this paper and is therefore not reflected in the table. It should be noted that small changes in how operations are or are not fused change entries in the table. What is important is that the table explains why the best blocked algorithms attain the performance that they attain.

which shows what computation would need to be performed if the update of A_{22} is delayed until the next iteration. Now, before $v_{21} = A_{22}^T u_{21}$ and $z_{21} = A_{22} u_{21}$ can be computed in the next iteration, $\textsc{Housev}(a_{21})$ has to be computed, which requires a_{21} to be updated. But what is important is that A_{22} can be updated by the two rank-1 updates from the previous iterations just before $v_{21} = A_{22}^T u_{21}$ and $w_{21} = A_{22} u_{21}$ are computed, which allows them to be "fused" into one operation that reads and writes A_{22} to and from memory only once. The algorithm in Figure 1 (right) takes advantage of these insights. To our knowledge it has not been previously published.

3.2 Lazy algorithm

We now show how the reduction to upper Hessenberg form can be restructured so that the update $A_{22} := A_{22} - (u_{21} y_{21}^T + z_{21} u_{21}^T)$ during each step can be avoided. This algorithm in and by itself is not practical, since (1) it requires too much temporary space, and (2) intermediate matrix-vector multiplications, which incur additional memory reads, eventually begin to dominate the operation. But it will become an integral part of the blocked algorithm discussed in Section 3.4. This algorithm was first reported in [9].

The rather curious choice of subscripts for u_{21}, and y_{21}, and z_{21} now becomes apparent: By passing

Algorithm: $[A, U, Y, Z] := \text{HessRed\_lazy\_unb}(A, U, Y, Z)$

Partition $X \rightarrow \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right)$
for $X \in \{A, U, Y, Z\}$
  where X_{TL} is 0×0
while $n(U_{TL}) < n(U)$ **do**
  Repartition

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

  for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Y, y, \psi), (Z, z, \zeta)\}$
    where χ_{11} is a scalar

$\alpha_{11} := \alpha_{11} - u_{10}^T y_{10} - z_{10}^T u_{10}$
$a_{21} := a_{21} - U_{20} y_{10} - Z_{20} u_{10}$
$a_{12}^T := a_{12}^T - u_{10}^T Y_{20}^T - z_{10}^T U_{20}^T$
$[u_{21}, \tau, a_{21}] := \text{Housev}(a_{21})$

$y_{21} := A_{22}^T u_{21}$
$z_{21} := A_{22} u_{21}$

$y_{21} := y_{21} - Y_{20}(U_{20}^T u_{21}) - U_{20}(Z_{20}^T u_{21})$
$z_{21} := z_{21} - U_{20}(Y_{20}^T u_{21}) - Z_{20}(U_{20}^T u_{21})$

$\beta := u_{21}^T z_{21}/2$
$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$
$z_{21} := (z_{21} - \beta u_{21}/\tau)/\tau$
$a_{12}^T := a_{12}^T - a_{12}^T u_{21} u_{21}^T/\tau$
$A_{02} := A_{02} - A_{02} u_{21} u_{21}^T/\tau$

  Continue with

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

  for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Y, y, \psi), (Z, z, \zeta)\}$
endwhile

Figure 3: Lazy unblocked algorithm for reduction to upper Hessenberg form.

matrices U, Y, and Z into the algorithm in Figure 1, and partitioning them just like we do A in that algorithm, we can accumulate the subvectors u_{21}, y_{21} and z_{21} into those matrices. Now, let us assume that at the top of the loop A_{BR} has not yet been updated. Then α_{11}, a_{21}, a_{12}^T and A_{22} have not yet been updated, which means we cannot perform many of the computations in the current iteration. However, if we let $\hat{\alpha}_{11}$, $\hat{a}_{21}$, $\hat{a}_{12}^T$, and $\hat{A}_{22}$ denote the original values in A in those locations, then the desired α_{11}, a_{21}, and a_{12}^T are given by

$$\begin{aligned} \alpha_{11} &= \hat{\alpha}_{11} - u_{10}^T y_{10} - z_{10}^T u_{10} \\ a_{21} &= \hat{a}_{21} - U_{20}^T y_{10} - Z_{20}^T u_{10} \\ a_{12}^T &= \hat{a}_{12}^T - u_{10}^T Y_{20}^T - z_{10}^T U_{20}^T \\ A_{22} &= \hat{A}_{22} - U_{20} Y_{20}^T - Z_{20} U_{20}^T. \end{aligned}$$

Thus, we start the iteration by updating in this fashion these parts of A.

Next, we observe that the updated A_{22} itself is not actually needed in updated form: We need to be able

Algorithm: $[A, U, Z, T] := \textsc{HessRed\_GQvdG\_unb}(b, A, U, Z, T)$

Partition $X \rightarrow \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right)$

for $X \in \{A, U, Z, T\}$

where X_{TL} is 0×0

while $n(U_{TL}) < b$ **do**

Repartition

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Z, z, \zeta), (T, t, \tau)\}$

where χ_{11} is a scalar

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) := \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) - \left(\begin{array}{c} Z_{00} \\ \hline z_{10}^T \\ \hline Z_{20} \end{array}\right) T_{00}^{-1} u_{10}$$

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-1} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T\right)^T \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right)$$

$[u_{21}, \tau_{11}, a_{21}] := \textsc{Housev}(a_{21})$

$$\left(\begin{array}{c} z_{01} \\ \hline \zeta_{11} \\ \hline z_{21} \end{array}\right) := \left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right) u_{21}$$

$t_{01} := U_{20}^T u_{21}$

Continue with

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Z, z, \zeta), (T, t, \tau)\}$

endwhile

Figure 4: GQvdG unblocked algorithm for the reduction to upper Hessenberg form.

to compute $A_{22}^T u_{21}$ and $A_{22} u_{21}$. But this can be done via the alternative computations

$$\begin{aligned} y_{21} &:= A_{22}^T u_{21} = \hat{A}_{22}^T u_{21} - Y_{20}(U_{20}^T u_{21}) - U_{20}(Z_{20}^T u_{21}) \\ z_{21} &:= A_{22} u_{21} = \hat{A}_{22} u_{21} - U_{20}(Y_{20}^T u_{21}) - Z_{20}(U_{20}^T u_{21}) \end{aligned}$$

which requires only matrix-vector multiplications. This inspires the algorithm in Figure 3.

3.3 GQvdG unblocked algorithm

The lazy algorithm discussed above requires at each step a matrix-vector and a transposed matrix-vector multiply which can be fused so that the matrix only needs to be brought into memory once. In this section, we show how the bulk of computation (and associated memory traffic) can be cast in terms of a single matrix multiplication per iteration with a much simpler algorithm that does not require fusing and thus no special implementation of the fused operation. This algorithm was first proposed by G. Quintana and van de Geijn in [19], which is why we call it the GQvdG unblocked algorithm. It is summarized in Figure 4.

The underlying idea builds upon how Householder transformations can be accumulated: The first b updates can be accumulated into a lower trapezoidal matrix U and upper triangular matrix T so that

$$\left(I - u_0 u_0^T/\tau_0\right)\left(I - u_1 u_1^T/\tau_1\right) \cdots \left(I - u_{b-1} u_{b-1}^T/\tau_{b-1}\right) = (I - U T^{-1} U^T).$$

After b iterations the basic unblocked algorithm overwrites matrix A with

$$\begin{aligned} A^{(b)} &= H(u_{b-1}) \cdots H(u_0) \hat{A} H(u_0) \cdots H(u_{b-1}) \\ &= \left(I - u_{b-1}u_{b-1}^T/\tau_{b-1}\right) \cdots \left(I - u_0 u_0^T/\tau_0\right) \hat{A} \left(I - u_0 u_0^T/\tau_0\right) \cdots H(u_{b-1}) \\ &= (I - UT^{-1}U^T)^T \hat{A} (I - UT^{-1}U^T) = (I - UT^{-1}U^T)^T (\hat{A} - \underbrace{\hat{A}U}_{Z} T^{-1}U^T) \\ &= (I - UT^{-1}U^T)^T (\hat{A} - ZT^{-1}U^T), \end{aligned}$$

where $\hat{A}$ denotes the original contents of A.

Let us assume that this process has proceeded for k iterations. Partition

$$X \rightarrow \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \text{ for } X \in \{A, \hat{A}, U, Z, T\},$$

where X_{TL} is $k \times k$. Then

$$A^{(k)} = \left(\begin{array}{c|c} A_{TL}^{(k)} & A_{TR}^{(k)} \\ \hline A_{BL}^{(k)} & A_{BR}^{(k)} \end{array}\right) =$$

$$\left(I - \left(\begin{array}{c} U_{TL} \\ \hline U_{BL} \end{array}\right) T_{TL}^{-1} \left(\begin{array}{c} U_{TL} \\ \hline U_{BL} \end{array}\right)^T\right)^T \left(\left(\begin{array}{c|c} \hat{A}_{TL} & \hat{A}_{TR} \\ \hline \hat{A}_{BL} & \hat{A}_{BR} \end{array}\right) - \left(\begin{array}{c} Z_{TL} \\ \hline Z_{BL} \end{array}\right) T_{TL}^{-1} \left(\begin{array}{c} U_{TL} \\ \hline U_{BL} \end{array}\right)^T\right).$$

Now, assume that after the first k iterations our algorithm leaves our variables in the following states:

- $A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$ contains $\left(\begin{array}{c|c} A_{TL}^{(k)} & \hat{A}_{TR} \\ \hline A_{BL}^{(k)} & \hat{A}_{BR} \end{array}\right)$. In other words, the first k columns have been updated and the rest of the columns are untouched.
- Only $\left(\begin{array}{c} U_{TL} \\ \hline U_{BR} \end{array}\right)$, T_{TL}, and $\left(\begin{array}{c} Z_{TR} \\ \hline Z_{BR} \end{array}\right)$ have been updated.

The question is how to advance the computation. Now, at the top of the loop, we expose

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (\hat{A}, \hat{a}, \hat{\alpha}), (U, u, \upsilon), (Z, z, \zeta), (T, t, \tau)$. In order to compute the next Householder transformation, the next column of A must be updated according to prior computation:

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) = \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-1} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T\right)^T \left(\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) - \underbrace{\left(\begin{array}{c} Z_{00} \\ \hline z_{10}^T \\ \hline Z_{20} \end{array}\right) T_{00}^{-1} u_{10}}_{\text{column } k \text{ of } Z_k T_k^{-1} U_k^T}\right),$$

which means first updating

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) := \left(\begin{array}{c} a_{01} - Z_{00} w_{10} \\ \hline \alpha_{11} - z_{10}^T w_{10} \\ \hline a_{21} - Z_{20} w_{10} \end{array}\right),$$

where $w_{10} = T_{00}^{-1} u_{10}$. Next, we need to perform the update

$$\left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-1} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T \right)^T \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right)$$

$$= \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) - \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right) T_{00}^{-T} \left(\begin{array}{c} U_{00} \\ \hline u_{10}^T \\ \hline U_{20} \end{array}\right)^T \left(\begin{array}{c} a_{01} \\ \hline \alpha_{11} \\ \hline a_{21} \end{array}\right) = \left(\begin{array}{c} a_{01} - U_{00} y_{10} \\ \hline \alpha_{11} - u_{10}^T y_{10} \\ \hline a_{21} - U_{20} y_{10} \end{array}\right),$$

where $y_{10} = T_{00}^{-T}(U_{00}^T a_{01} + u_{10}\alpha_{11} + U_{20}^T a_{21})$. After these computations we can compute the next Householder transform from a_{21}, updating a_{21}:

- $[u_{21}, \tau, a_{21}] := \text{HOUSEV}(a_{21})$.

The next column of Z is computed by

$$\left(\begin{array}{c} z_{01} \\ \hline \zeta_{11} \\ \hline z_{21} \end{array}\right) := \left(\begin{array}{c|c|c} \hat{A}_{00} & \hat{a}_{01} & \hat{A}_{02} \\ \hline \hat{a}_{10}^T & \hat{\alpha}_{11} & \hat{a}_{12}^T \\ \hline \hat{A}_{20} & \hat{a}_{21} & \hat{A}_{22} \end{array}\right) \left(\begin{array}{c} 0 \\ \hline 0 \\ \hline u_{21} \end{array}\right) = \left(\begin{array}{c} \hat{A}_{02} u_{21} \\ \hline \hat{a}_{12}^T u_{21} \\ \hline \hat{A}_{22} u_{21} \end{array}\right).$$

We finish by computing the next column of T:

$$\left(\begin{array}{c|c|c} T_{00} & \hat{t}_{01} & \hat{T}_{02} \\ \hline 0 & \hat{\tau}_{11} & \hat{t}_{12}^T \\ \hline 0 & 0 & \hat{T}_{22} \end{array}\right) := \left(\begin{array}{c|c|c} T_{00} & U_{20}^T u_{21} & \hat{T}_{02} \\ \hline 0 & \frac{1}{2} u_{21}^T u_{21} & \hat{t}_{12}^T \\ \hline 0 & 0 & \hat{T}_{22} \end{array}\right).$$

Note that $\frac{1}{2} u_{21}^T u_{21}$ is equal to the τ computed by $\text{HOUSEV}(a_{21})$, and thus it need not be recomputed to update τ_{11}.

3.4 Blocked algorithms

We now discuss how much of the computation can be cast in terms of matrix-matrix multiplication. The first such blocked algorithm was reported in [9]. That algorithm corresponds roughly to our blocked Algorithm 1.

In Figure 5 we give four blocked algorithms which differ by how computation is accumulated in the body of the loop:

- Two correspond to using the unblocked algorithms in Figure 1.
- A third results from using the lazy algorithm in Figure 3. For this variant, we introduce matrices U, Y, and Z of width b in which vectors computed by the lazy unblocked algorithm are accumulated. We are not aware of this algorithm having been reported before.
- The fourth results from using the algorithm in Figure 4. It returns matrices U, Z, and T. It was first reported in [19] and we will call it the GQvdG blocked algorithm.

Let us consider having progressed through the matrix so that it is in the state

$$A = \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), \quad U = \left(\begin{array}{c} U_T \\ \hline U_B \end{array}\right), \quad Y = \left(\begin{array}{c} Y_T \\ \hline Y_B \end{array}\right), \quad Z = \left(\begin{array}{c} Z_T \\ \hline Z_B \end{array}\right),$$

where A_{TL} is $b \times b$. Assume that the factorization has completed with A_{TL} and A_{BL} (meaning that A_{TL} is upper Hessenberg and A_{BL} is zero except for its top-right most element), and A_{TR} and A_{BR} have been updated so that only an upper Hessenberg factorization of A_{BR} has to be completed, updating the A_{TR} submatrix correspondingly. In the next iteration of the blocked algorithm, we perform the following steps:

Algorithm: $[A] := \text{HessRed\_blk}(A, T)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $X \rightarrow \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right)$

for $X \in \{T, U, Y, Z\}$

where A_{TL} is 0×0 and T_T, U_T, Y_T, and Z_T have 0 rows

while $m(A_{TL}) < m(A)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline X_2 \end{array}\right)$$

for $X \in \{T, U, Y, Z\}$

where A_{11} is $b \times b$ and T_1, U_1, Y_1, and Z_1 have b rows

---

Algorithm 1, 2: (blocked + basic unblocked, blocked + rearranged unblocked)

$[A_{BR}, U_B] := \text{HessRed\_unb}(b, A_{BR})$

$T_1 = \frac{1}{2}D + S$ where $U_B^T U_B = S^T + D + S$

$A_{TR} := A_{TR}(I - U_B T_1^{-1} U_B^T)$

Algorithm 3: (blocked + lazy unblocked)

$[A_{BR}, U_B, Y_B, Z_B] := \text{HessRed\_lazy\_unb}(b, A_{BR}, U_B, Y_B, Z_B)$

$T_1 = \frac{1}{2}D + S$ where $U_B^T U_B = S^T + D + S$

$A_{TR} := A_{TR}(I - U_B T_1^{-1} U_B^T)$

$A_{22} := A_{22} - U_2 Y_2^T - Z_2 U_2^T$

Algorithm 4: (GQvdG blocked + GQvdG unblocked)

$[A_{BR}, U_B, Z_B, T_1] := \text{HessRed\_GQvdG\_unb}(b, A_{BR}, U_B, Z_B, T_1)$

$A_{TR} := A_{TR}(I - U_B T_1^{-1} U_B^T)$

$$\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(I - \left(\begin{array}{c} U_1 \\ \hline U_2 \end{array}\right) T_1^{-1} \left(\begin{array}{c} U_1 \\ \hline U_2 \end{array}\right)^T\right)^T \left(\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) - \left(\begin{array}{c} Z_1 \\ \hline Z_2 \end{array}\right) T_1^{-1} U_2^T\right)$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \leftarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline X_2 \end{array}\right)$$

for $X \in \{T, U, Y, Z\}$

endwhile

Figure 5: Blocked reduction to Hessenberg form based on original or rearranged algorithm. The call to HessRed_unb performs the first b iterations of one of the unblocked algorithms in Figures 1 or 3. In the case of the algorithms in Figure 1, U_B accumulates and returns the vectors u_{21} encountered in the computation and Y_B and Z_B are not used.

- Perform the first b iterations of the lazy algorithm with matrix A_{BR}, accumulating the appropriate vectors in U_B, Y_B, and Z_B.
- Apply the resulting Householder transformations from the right to A_{TR}. In Section 2.3 we discussed that this requires the computation of $U^T U = S^T + D + S$, where D and S equal the diagonal and strictly upper triangular part of $U^T U$, after which $A_{TR} := A_{TR}(I - UT^{-1}U^T) = A_{TR} - A_{TR}UT^{-1}U^T$ with $T = \frac{1}{2}D + S$.
- Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \quad \left(\begin{array}{c} U_T \\ \hline U_B \end{array}\right) \rightarrow \left(\begin{array}{c} U_0 \\ \hline U_1 \\ \hline U_2 \end{array}\right), \quad \ldots$$

- Update $A_{22} := A_{22} - U_2 Y_2^T - Z_2 U_2^T$.

- Move the thick line (which denotes how far the factorization has proceeded) forward by the block size:

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \quad \left(\begin{array}{c} U_T \\ \hline U_B \end{array}\right) \leftarrow \left(\begin{array}{c} U_0 \\ \hline U_1 \\ \hline U_2 \end{array}\right), \quad \ldots$$

Proceeding like this block-by-block computes the reduction to upper Hessenberg form while reducing the size of the matrices U, Y, and Z, casting some of the computation in terms of matrix-matrix multiplications that are known to achieve high performance.

When one of the unblocked algorithms in Figure 1 is used instead, A_{22} is already updated upon return from HessRed_unb and thus only the update of A_{TR} can be accelerated by calls to level-3 BLAS operations.

The GQvdG blocked algorithm, which uses the GQvdG unblocked algorithm, was incorporated into recent releases of LAPACK, modulo a small change that accumulates T^{-1} instead of T. Prior to this, an algorithm that used the lazy unblocked algorithm but also updated A_{TR} as part of that unblocked algorithm (and thus cast less computation in terms of level-3 BLAS) was part of LAPACK [9]. A comparison between the GQvdG blocked algorithm and this previously used algorithm can be found in [19].

3.5 Fusing operations

We now discuss how three sets of operations encountered in the various algorithms can be fused to reduce memory traffic.

In the lazy algorithm, delaying the update of A_{22} yields the following three operations that can be fused (here we drop the subscripts):

$$\begin{array}{l} A := A - (uy^T + zu^T) \\ v := A^T x \\ w := Ax \end{array}$$

Partition

$$A \rightarrow \left(a_0 | \cdots | a_{n-1}\right), \quad u \rightarrow \begin{pmatrix} \upsilon_0 \\ \vdots \\ \upsilon_{n-1} \end{pmatrix}, \quad v \rightarrow \begin{pmatrix} \nu_0 \\ \vdots \\ \nu_{n-1} \end{pmatrix}, \quad x \rightarrow \begin{pmatrix} \chi_0 \\ \vdots \\ \chi_{n-1} \end{pmatrix}, \quad y \rightarrow \begin{pmatrix} \psi_0 \\ \vdots \\ \psi_{n-1} \end{pmatrix}.$$

Then the following steps, $0 \leq i < n$, compute the desired result (provided initially $w = 0$):

$$a_i := a_i - \psi_i u - \upsilon_i z; \; \nu_i := a_i^T x; \; w := w + \chi_i a_i.$$

Similarly,

$$\begin{array}{l} v := A^T x \\ w := Ax \end{array}$$

can be computed via

$$\nu_i := a_i^T x; \; w := w + \chi_i a_i, \quad 0 \leq i < n.$$

Finally,

$$\begin{array}{l} y := y - Y(U^T u) - U(Z^T u) \\ z := z - U(Y^T u) - Z(U^T u) \end{array}$$

can be computed by partitioning

$$U \rightarrow \left(\begin{array}{c|c|c} u_0 & \cdots & u_{k-1} \end{array}\right), \quad Y \rightarrow \left(\begin{array}{c|c|c} y_0 & \cdots & y_{k-1} \end{array}\right), \quad Z \rightarrow \left(\begin{array}{c|c|c} z_0 & \cdots & z_{k-1} \end{array}\right),$$

and computing

$$\alpha := u_i^T u; \; \beta := z_i^T u; \; \gamma := y_i^T u; \; y := y - \alpha y_i - \beta u_i; \; z := z - \alpha z_i - \gamma u_i,$$

for $0 \leq i < n$.

Algorithm: $[A] := \textsc{TriRed\_unb}(b, A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right)$

for $x \in \{u, y\}$

  where A_{TL} is 0×0 and u_T, y_T have 0 rows

while $m(A_{TL}) < m(A)$ **do**

  Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array}\right)$$

  for $(x, \chi) \in \{(u, \upsilon), (y, \psi)\}$

   where α_{11}, υ_{11}, and ψ_{11} are scalars

Basic unblocked:	Rearranged unblocked:	
	$\alpha_{11} := \alpha_{11} - 2\upsilon_{11}\psi_{11}$	$(\star)$
	$a_{21} := a_{21} - (u_{21}\psi_{11} + y_{21}\upsilon_{11})$	$(\star)$
$[u_{21}, \tau, a_{21}] := \textsc{Housev}(a_{21})$	$[x_{21}, \tau, a_{21}] := \textsc{Housev}(a_{21})$	
	$A_{22} := A_{22} - u_{21}y_{21}^T - y_{21}u_{21}^T$	$(\star)$
$y_{21} := A_{22}u_{21}$	$v_{21} := A_{22}x_{21}$	
	$u_{21} := x_{21}$; $y_{21} := v_{21}$	
$\beta := u_{21}^T y_{21}/2$	$\beta := u_{21}^T y_{21}/2$	
$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$	$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$	
$A_{22} := A_{22} - u_{21}y_{21}^T - y_{21}u_{21}^T$		

  Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array}\right)$$

  for $(x, \chi) \in \{(u, \upsilon), (y, \psi)\}$

endwhile

Figure 6: Unblocked algorithms for reduction to tridiagonal form. Left: basic algorithm. Right: rearranged to allow fusing of operations. Operations marked with $(\star)$ are not executed during the first iteration.

4 Reduction to tridiagonal form

The first step towards computing the eigenvalue decomposition of a symmetric matrix is to reduce the matrix to tridiagonal form.

Let $A \in \mathbb{R}^{n \times n}$ be symmetric. If $A \rightarrow QBQ^T$ where B is upper Hessenberg and Q is orthogonal, then B is symmetric and therefore tridiagonal. In this section we show how to take advantage of symmetry, assuming that matrix A is stored in only the lower triangular part of A and only the lower triangular part of that matrix is overwritten with B.

When matrix A is symmetric, and only the lower triangular part is stored and updated, the unblocked algorithms for reducing A to upper Hessenberg form can be changed by noting that $v_{21} = w_{21}$ and $y_{21} = z_{21}$. This motivates the algorithms in Figures 6–8. The blocked algorithm and associated unblocked algorithm was first reported in [9].

In the rearranged algorithm, delaying the update of A_{22} allows the highlighted operations in Figure 6 (right) to be fused via the algorithm in Figure 9. We leave it as an exercise to the reader to fuse the highlighted operations in Figure 7.

Algorithm: $[A, U, Y] := \text{TRIRED\_LAZY\_UNB}(A, U, Y)$

Partition $X \rightarrow \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right)$

for $X \in \{A, U, Y\}$

where X_{TL} is 0×0

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

Repartition

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Y, y, \psi)\}$

where χ_{11} is a scalar

$\alpha_{11} := \alpha_{11} - u_{10}^T y_{10} - y_{10}^T u_{10}$

$a_{21} := a_{21} - U_{20} y_{10} - Y_{20} u_{10}$

$[u_{21}, \tau, a_{21}] := \text{HOUSEV}(a_{21})$

$y_{21} := A_{22} u_{21}$

$y_{21} := y_{21} - Y_{20}(U_{20}^T u_{21}) - U_{20}(Y_{20}^T u_{21})$

$\beta := u_{21}^T y_{21}/2$

$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$

Continue with

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array}\right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Y, y, \psi)\}$

endwhile

Figure 7: Lazy unblocked reduction to tridiagonal form.

5 Reduction to bidiagonal form

The previous sections were inspired by the paper [12] that discusses how fused operations can benefit algorithms for the reduction of a matrix to bidiagonal form. The purpose of this section is to present the basic and rearranged unblocked algorithms for this operation with our notation to facilitate the comparing and contrasting of the reduction to upper Hessenberg and tridiagonal form algorithms to those for the reduction to bidiagonal form.

The first step towards computing the Singular Value Decomposition (SVD) of $A \in \mathbb{R}^{m \times n}$ is to reduce the matrix to bidiagonal form: $A \rightarrow UBV^T$ where B is a bidiagonal matrix (nonzero diagonal and superdiagonal) and U and V are again square and orthogonal.

For simplicity, we explain the algorithms for the case where A is square.

5.1 Basic algorithm

The basic algorithm for this operation, overwriting A with the result B, can be explained as follows:

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right)$.

- Let $\left[\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array}\right)\right] := \text{HOUSEV}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)\right)$.[4]

[4]Note that the semantics here indicate that α_{11} is overwritten by the first element of $\left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array}\right)$.

$$
\begin{array}{|l|}
\hline
\textbf{Algorithm: } [A,U,Y] := \textsc{TriRed\_blk}(A,U,Y) \\
\hline
\textbf{Partition } A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), X \rightarrow \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \\
\quad \textbf{for } X \in \{U, Y\} \\
\qquad \textbf{where } A_{TL} \text{ is } 0 \times 0 \text{ and } U_T,\ Y_T \text{ have 0 rows} \\
\textbf{while } m(A_{TL}) < m(A) \textbf{ do} \\
\qquad \textbf{Determine block size } b \\
\qquad \textbf{Repartition} \\
\qquad\quad \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline X_2 \end{array}\right) \\
\qquad\quad \textbf{for } X \in \{U, Y\} \\
\qquad\qquad \textbf{where } A_{11} \text{ is } b \times b \text{ and } U_1, \text{ and } Y_1 \text{ have } b \text{ rows} \\
\hline\hline
\qquad [A_{BR}, U_B, Y_B] := \textsc{TriRed\_lazy\_unb}(b, A_{BR}, U_B, Y_B) \\
\qquad A_{22} := A_{22} - U_2 Y_2^T - Y_2 U_2^T \\
\hline\hline
\qquad \textbf{Continue with} \\
\qquad\quad \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \leftarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline X_2 \end{array}\right) \\
\qquad\quad \textbf{for } X \in \{U, Y\} \\
\textbf{endwhile} \\
\hline
\end{array}
$$

Figure 8: Blocked reduction to tridiagonal form based on original or rearranged algorithm. TriRed_unb performs the first b iterations of the lazy unblocked algorithm in Figure 7.

- Update

$$
\begin{aligned}
\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) &:= \left(I - \left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)^T / \tau_L\right)\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) \\
&= \left(\begin{array}{c|c} \alpha - \psi_{11}/\tau_L & a_{12}^T - y_{21}^T/\tau_L \\ \hline 0 & A_{22} - u_{21} y_{21}^T/\tau_L \end{array}\right),
\end{aligned}
$$

  where $\psi_{11} = \alpha_{11} + u_{21}^T a_{21}$ and $y_{21}^T = a_{12}^T + u_{21}^T A_{22}$. Note that $\alpha_{11} := \alpha - \psi_{11}/\tau_L$ need not be executed since this update was performed by the instance of Housev above.

- Let $[v_{21}, \tau_R, a_{12}] := \textsc{Housev}(a_{12})$.

- Update $A_{22} := A_{22}(I - v_{21} v_{21}^T/\tau_R) = A_{22} - z_{21} v_{21}^T/\tau_R$, where $z_{21} = A_{22} v_{21}$.

- Continue this process with the updated A_{22}.

The resulting algorithm, slightly rearranged, is given in Figure 10 (left).

5.2 Rearranged algorithm

We now show how, again, the loop can be restructured so that multiple updates of, and multiplications with, A_{22} can be fused. Focus on the update $A_{22} := A_{22} - (u_{21} y_{21}^T + z_{21} v_{21}^T)$. Partition

$$
A_{22} \rightarrow \left(\begin{array}{c|c} \alpha_{11}^+ & a_{12}^{+T} \\ \hline a_{21}^+ & A_{22}^+ \end{array}\right), \quad u_{21} \rightarrow \left(\begin{array}{c} v_{11}^+ \\ \hline u_{21}^+ \end{array}\right), \quad y_{21} \rightarrow \left(\begin{array}{c} \psi_{11}^+ \\ \hline y_{21}^+ \end{array}\right), \quad z_{21} \rightarrow \left(\begin{array}{c} \zeta_{11}^+ \\ \hline z_{21}^+ \end{array}\right), \quad v_{21} \rightarrow \left(\begin{array}{c} \nu_{11}^+ \\ \hline v_{21}^+ \end{array}\right),
$$

Algorithm: $[A] := \textsc{Fused\_Syr2\_Symv}(A, x, y, u, v)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$

for $z \in \{x, y, u, v\}$

  where A_{TL} is 0×0, x_T, y_T, u_T, v_T have 0 elements

  $v = 0$

while $m(A_{TL}) < m(A)$ **do**

  Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ \hline z_2 \end{array}\right)$$

  for $(z, \zeta) \in \{(x, \chi), (y, \psi), (u, \upsilon), (v, \nu)\}$

   where α_{11}, χ_1, ψ_1, υ_1, ν_1 are scalars

$$\left.\begin{array}{ll} \alpha_{11} := \alpha_{11} + 2\psi_1\upsilon_1 & \\ a_{21} := a_{21} + \psi_1 u_2 + \upsilon_1 y_2 & (\textsc{axpy} \times 2) \end{array}\right\} \begin{array}{l} \text{toward} \\ A := A + (uy^T + yu^T) \end{array}$$

$$\left.\begin{array}{ll} \nu_1 := \nu_1 + \alpha_{11}\chi_1 + a_{21}^T x_2 & (\textsc{dot}) \\ v_2 := v_2 + \chi_1 a_{21} & (\textsc{axpy}) \end{array}\right\} \begin{array}{l} \text{toward} \\ v := Ax \end{array}$$

  Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_0 \\ \hline \zeta_1 \\ \hline z_2 \end{array}\right)$$

  for $(z, \zeta) \in \{(x, \chi), (y, \psi), (u, \upsilon), (v, \nu)\}$

endwhile

Figure 9: Algorithm that fuses a symmetric rank-2 update and a symmetric matrix-vector multiply: $A := A + (uy^T + yu^T)$; $v := Ax$, where A is symmetric and stored in the lower triangular part of A.

where + indicates the partitioning in the next iteration. Then

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11}^+ & a_{12}^{+T} \\ \hline a_{21}^+ & A_{22}^+ \end{array}\right) &:= \left(\begin{array}{c|c} \alpha_{11}^+ & a_{12}^{+T} \\ \hline a_{21}^+ & A_{22}^+ \end{array}\right) - \left(\begin{array}{c} \upsilon_{11}^+ \\ \hline u_{21}^+ \end{array}\right)\left(\begin{array}{c} \psi_{11}^+ \\ \hline y_{21}^+ \end{array}\right)^T - \left(\begin{array}{c} \zeta_{11}^+ \\ \hline z_{21}^+ \end{array}\right)\left(\begin{array}{c} \nu_{11}^+ \\ \hline v_{21}^+ \end{array}\right)^T \\ &= \left(\begin{array}{c|c} \alpha_{11}^+ - \upsilon_{11}^+\psi_{11}^+ - \zeta_{11}^+\nu_{11}^+ & a_{12}^{+T} - \upsilon_{11}^+ y_{21}^{+T} - \zeta_{11}^+ v_{21}^{+T} \\ \hline a_{21}^+ - u_{21}^+\psi_{11}^+ - z_{21}^+\nu_{11}^+ & A_{22}^+ - u_{21}^+ y_{21}^{+T} - z_{21}^+ v_{21}^{+T} \end{array}\right), \end{aligned}$$

which shows how the update of A_{22} can be delayed until the next iteration. If $u_{21} = y_{21} = z_{21} = v_{21} = 0$ during the first iteration, the body of the loop may be changed to

$$\begin{array}{l} \alpha_{11} := \alpha_{11} - \upsilon_{11}\psi_{11} - \zeta_{11}\nu_{11} \\ a_{21} := a_{21} - u_{21}\psi_{11} - z_{21}\nu_{11} \\ a_{12}^T := a_{12}^T - \upsilon_{11} y_{21}^T - \zeta_{11} v_{21}^T \\ \left[\left(\begin{array}{c} 1 \\ \hline u_{21}^+ \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array}\right)\right] := \textsc{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)\right) \\ A_{22} := A_{22} - u_{21} y_{21}^T - z_{21} v_{21}^T \\ y_{21} := a_{12} + A_{22}^T u_{21}^+ \\ a_{12}^T := a_{12}^T - y_{21}^T / \tau_L \\ [v_{21}, \tau_R, a_{12}] := \textsc{Housev}\,(a_{12}) \\ \beta := y_{21}^T v_{21} \\ y_{21} := y_{21} / \tau_L \\ z_{21} := (A_{22} v_{21} - \beta u_{21}^+ / \tau_L) / \tau_R \end{array}$$

Now, the goal becomes to bring the three highlighted updates together. The problem is that the last update, which requires v_{21}, cannot commence until after the second call to Housev completes. This dependency

Algorithm: $[A] := \textsc{BiRed\_unb}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right)$

for $x \in \{u, v, y, z\}$

where A_{TL} is 0×0, u_T, v_T, y_T, z_T have 0 elements

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array}\right)$$

for $(x, \chi) \in \{(u, \upsilon), (v, \nu), (y, \psi), (z, \zeta)\}$

where α_{11}, υ_{11}, ν_{11}, ψ_{11}, and ζ_{11} are scalars

Basic unblocked:

$\left[\left(\begin{array}{c} 1 \\ u_{21} \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ 0 \end{array}\right)\right] := \textsc{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array}\right)\right)$

$y_{21} := a_{12} + A_{22}^T u_{21}$
$a_{12}^T := a_{12}^T - y_{21}^T / \tau_L$

$[v_{21}, \tau_R, a_{12}] := \textsc{Housev}(a_{12})$

$\beta := y_{21}^T v_{21}$
$y_{21} := y_{21} / \tau_L$
$z_{21} := (A_{22} v_{21} - \beta u_{21} / \tau_L) / \tau_R$

$A_{22} := A_{22} - u_{21} y_{21}^T - z_{21} v_{21}^T$

Rearranged unblocked:

$\alpha_{11} := \alpha_{11} - \upsilon_{11}\psi_{11} - \zeta_{11}\nu_{11}$ $(\star)$
$a_{21} := a_{21} - u_{21}\psi_{11} - z_{21}\nu_{11}$ $(\star)$
$a_{12}^T := a_{12}^T - \upsilon_{11} y_{21}^T - \zeta_{11} v_{21}^T$ $(\star)$

$\left[\left(\begin{array}{c} 1 \\ u_{21}^+ \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ 0 \end{array}\right)\right] := \textsc{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ a_{21} \end{array}\right)\right)$

$a_{12}^+ := a_{12} - a_{12} / \tau_L$
$A_{22} := A_{22} - u_{21} y_{21}^T - z_{21} v_{21}^T$ $(\star)$
$y_{21} := A_{22}^T u_{21}^+$
$a_{12}^+ := a_{12}^+ - y_{21} / \tau_L$
$w_{21} := A_{22} a_{12}^+$
$y_{21} := y_{21} + a_{12}$
$[\psi_{11} - \alpha_{12}, \tau_R, \alpha_{12}] := \textsc{Houses}(a_{12}^+)$
$v_{21} := (a_{12}^+ - \alpha_{12} e_0) / (\psi_{11} - \alpha_{12})$;
$a_{12}^T := \alpha_{12} e_0^T$
$u_{21} := u_{21}^+$
$\beta := y_{21}^T v_{21}$
$y_{21} := y_{21} / \tau_L$
$z_{21} := (w_{21} - \alpha_{12} A_{22} e_0) / (\psi_{11} - \alpha_{12})$
$z_{21} := z_{21} - \beta u_{21} / \tau_L$
$z_{21} := z_{21} / \tau_R$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array}\right)$$

for $(x, \chi) \in \{(u, \upsilon), (v, \nu), (y, \psi), (z, \zeta)\}$

endwhile

Figure 10: Unblocked algorithms for reduction to bidiagonal form. Left: basic algorithm. Right: rearranged to allow fusing of operations. Operations marked with $(\star)$ are not executed during the first iteration.

can be circumvented by observing that one can perform a matrix-vector multiply of A_{22} with the vector $a_{12}^T = a_{12}^T - y_{21}^T/\tau_L$ instead of with v_{21}, after which the result can be updated as if the multiplication had used the output of the Housev, as indicated by Eq. (3) in Section 2. These observations justify the rearrangement of the computations as indicated in Figure 10 (right).

5.3 Lazy algorithms

A lazy algorithm can be derived by not updating A_{22} at all, and instead accumulating the updates in matrix U, V, Y, and Z, much like was done for the other reduction to condensed form operations.

We start with the rearranged algorithm to make sure that

$$\begin{array}{l} y_{21} := A_{22}^T u_{21}^+ \\ a_{12}^+ := a_{12}^+ - y_{21}/\tau_L \\ w_{21} := A_{22} a_{12}^+ \end{array}$$

can still be fused. Next, the key is to realize that what was previously a multiplication by A_{22} must now be replaced by a multiplication by $A_{22} - U_{20}Y_{20}^T - Z_{20}V_{20}^T$. This yields the algorithm in Figure 11 (right) which was first proposed by Howell et al. [12].

For completeness, we include in Figure 11 (left) a basic algorithm which does not rearrange operations for fusing, but still has the "lazy" property whereby A_{22} is never updated.

5.4 Blocked algorithms

Finally, a blocked algorithm is given in Figure 12. The basic lazy unblocked algorithm in conjunction with the blocked algorithm was first published in [9] and is part of LAPACK. The rearranged lazy unblocked algorithm in conjunction with the blocked algorithm (Howell's Algorithm) was proposed by Howell et al. and published in [12].

5.5 Fusing operations

Once again, we leave it as an exercise to the reader to construct loop-based fusings of the operations highlighted in Figures 10 and 11.

6 Impact on performance

We now report performance for implementations of various algorithms that is attained in practice. We stress that final conclusions cannot be made until someone (not us) fully optimizes the fused operations.

6.1 Platform details

All experiments were performed on a single core of a Dell PowerEdge R900 server consisting of four Intel "Dunnington" six-core processors. Each core provides a peak performance of 10.64 GFLOPS. Performance experiments were gathered under the GNU/Linux 2.6.18 operating system. Source code was compiled by the Intel C/C++ Compiler, version 11.1. All experiments were performed in double precision floating-point arithmetic on real-valued matrices.

All reduction to condensed form implementations reported in this paper were linked to the BLAS provided by GotoBLAS2 1.10. All LAPACK implementations were obtained via the netlib distribution of LAPACK version 3.2.1. For the reduction to bidiagonal form we also compare against an implementation by Howell published in [12] and available from [11].

In many of our papers, the top line of a graph represents peak attainable performance. But given that the reduction algorithms cannot be expected to attain near-peak performance (since inherently a significant fraction of computation is in memory-intensive level-2 BLAS operations), we do not follow that convention in this paper so as to make the very busy graphs more readable.

Algorithm: $[A, U, V, Y, Z] := \text{BiRed\_lazy\_unb}(A, U, V, Y, Z)$

Partition $X \rightarrow \left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array} \right)$

for $X \in \{A, U, V, Y, Z\}$

where X_{TL} is 0×0

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

Repartition

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array} \right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Y, y, \psi), (Z, z, \zeta)\}$

where χ_{11} is a scalar

Lazy basic unblocked:

$\alpha_{11} := \alpha_{11} - u_{10}^T y_{10} - z_{10}^T v_{10}$

$a_{21} := a_{21} - U_{20} y_{10} - Z_{20} v_{10}$

$a_{12}^T := a_{12}^T - u_{10}^T Y_{20}^T - z_{10}^T V_{20}^T$

$\left[\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array} \right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array} \right) \right] := \text{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array} \right) \right)$

$y_{21} := a_{12} + A_{22}^T u_{21}$
$\quad - Y_{20} U_{20}^T u_{21} - V_{20} Z_{20}^T u_{21}$

$a_{12}^T := a_{12}^T - y_{21}^T / \tau_L$

$[v_{21}, \tau_R, a_{12}] := \text{Housev}\,(a_{12})$

$\beta := y_{21}^T v_{21}$

$y_{21} := y_{21} / \tau_L$

$z_{21} := (A_{22} v_{21}$
$\quad - U_{20} Y_{20}^T v_{21} - Z_{20} V_{20}^T v_{21}$
$\quad - \beta u_{21} / \tau_L) / \tau_R$

Lazy rearranged (Howell) unblocked:

$\alpha_{11} := \alpha_{11} - u_{10}^T y_{10} - z_{10}^T v_{10}$

$a_{21} := a_{21} - U_{20} y_{10} - Z_{20} v_{10}$

$a_{12}^T := a_{12}^T - u_{10}^T Y_{20}^T - z_{10}^T V_{20}^T$

$\left[\left(\begin{array}{c} 1 \\ \hline u_{21}^+ \end{array} \right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array} \right) \right] := \text{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array} \right) \right)$

$a_{12}^+ := a_{12} - a_{12} / \tau_L$

$y_{21} := -Y_{20} U_{20}^T u_{21}^+ - V_{20} Z_{20}^T u_{21}^+$

$y_{21} := y_{21} + A_{22}^T u_{21}^+$

$a_{12}^+ := a_{12}^+ - y_{21} / \tau_L$

$w_{21} := A_{22} a_{12}^+$

$w_{21} := w_{21} - U_{20} Y_{20}^T a_{12}^+ - Z_{20} V_{20}^T a_{12}^+$

$a_{22l} := A_{22} e_0 - U_{20} Y_{20}^T e_0 - Z_{20} V_{20}^T e_0$

$y_{21} := a_{12} + y_{21}$

$[\psi_{11} - \alpha_{12}, \tau_R, \alpha_{12}] := \text{Houses}(a_{12}^+)$

$v_{21} := (a_{12}^+ - \alpha_{12} e_0) / (\psi_{11} - \alpha_{12});$

$a_{12}^T := \alpha_{12} e_0^T$

$u_{21} := u_{21}^+$

$\beta := y_{21}^T v_{21}$

$y_{21} := y_{21} / \tau_L$

$z_{21} := (w_{21} - \alpha_{12} a_{22l}) / (\psi_{11} - \alpha_{12})$

$z_{21} := z_{21} - \beta u_{21} / \tau_L$

$z_{21} := z_{21} / \tau_R$

Continue with

$$\left(\begin{array}{c|c} X_{TL} & X_{TR} \\ \hline X_{BL} & X_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} X_{00} & x_{01} & X_{02} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T \\ \hline X_{20} & x_{21} & X_{22} \end{array} \right)$$

for $(X, x, \chi) \in \{(A, a, \alpha), (U, u, \upsilon), (Y, y, \psi), (Z, z, \zeta)\}$

endwhile

Figure 11: Lazy versions of the algorithm in Figure 10. Upon entry matrix A is $n \times n$ and matrices U, V, Y, and Z are $n \times b$. Note that the multiplications $A_{22} e_0$, $Y_{20}^T e_0$, and $U_{20}^T e_0$ do not require computation: they simply extract the first column or row of the given matrix.

Algorithm: $[A] := \textsc{BiRed\_blk}(A, U, V, Y, Z)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $X \rightarrow \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right)$

for $X \in \{U, V, Y, Z\}$

where A_{TL} is 0×0 and U_T, V_T, Y_T, Z_T have 0 rows

while $m(A_{TL}) < m(A)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline X_2 \end{array}\right)$$

for $X \in \{U, V, Y, Z\}$

where A_{11} is $b \times b$ and U_1, V_1, Y_1, and Z_1 have b rows

---

$[A_{BR}, U_B, V_B, Y_B, Z_B] := \textsc{BiRed\_lazy\_unb}(b, A_{BR}, U_B, V_B, Y_B, Z_B)$

$A_{22} := A_{22} - U_2 Y_2^T - Z_2 V_2^T$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \leftarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ \hline X_2 \end{array}\right)$$

for $X \in \{U, V, Y, Z\}$

endwhile

Figure 12: Blocked algorithm for reduction to bidiagonal form. For simplicity, it is assumed that A is $n \times n$ where n is an integer multiple of b. Matrices U, V, Y, and Z are all $n \times b$.

6.2 Fused operation implementations

Fused operations were coded in terms of level-1 BLAS. Ideally, they would be coded at the same level of abstraction as highly optimized level-2 BLAS, which often means in assembly code. We do not have the expertise to do this ourselves. Thus, regardless of the performance observed using these fused operations, we suspect that higher performance may be attainable provided that the fused operations are carefully coded by an expert. The "Build To Order BLAS" project [21] studies the systematic and automatic optimization of these kinds of fused operations and some such fused operations are available as part of vendor libraries.

6.3 Implementations of the reduction algorithms

The algorithms were implemented using the FLAME/C API [24, 3] which allows the implementations to closely mirror the algorithms presented in this paper. Since this API carries considerable overhead that affects performance, the unblocked algorithms were translated into lower-level (LAPACK-like) implementations that use the BLAS-like Interface Subprograms (BLIS) interface [26]. This is a C interface that resembles the BLAS interface but is more natural for C and fixes certain problems for the routines that compute with (single and double precision) complex datatypes. All these implementations are part of the standard `libflame` distribution so that others can experiment with further optimizations.

6.4 Tuning of block size

We performed experiments to determine the optimal block size for the blocked algorithms. A block size of 32, the default block size for the LAPACK implementation, appeared to be near-optimal and was used for all experiments.

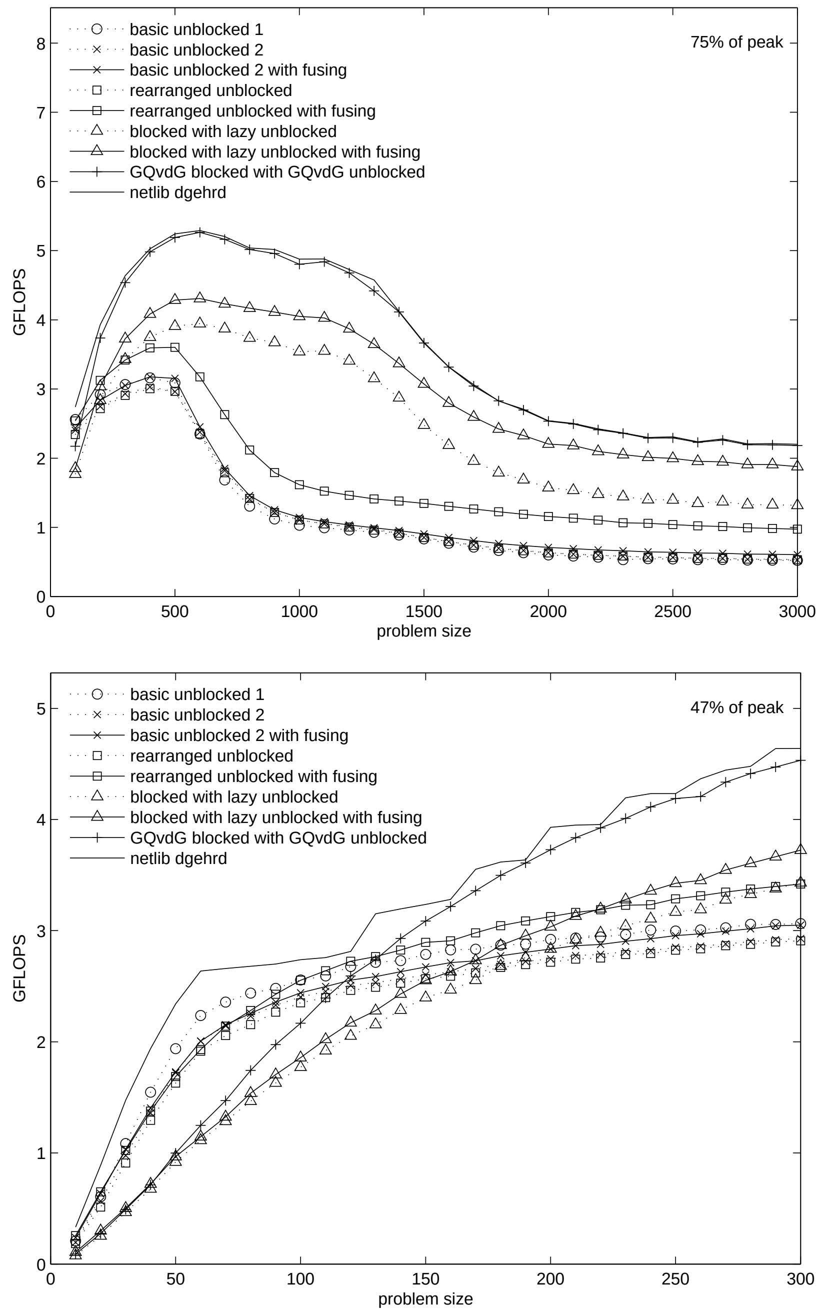

Figure 13: Performance of various implementations of reduction to upper Hessenberg form for problem sizes up to 3000 (top) and up to 300 (bottom). Implementations of blocked algorithms use a block size of 32.

6.5 Reduction to upper Hessenberg form

Performance of the various implementations of reduction to upper Hessenberg form are given in Figure 13. The netlib LAPACK algorithm always outperforms the other implementations, although eventually the "GQvdG blocked with GQvdG unblocked" algorithm catches up. Note that netlib `dgehrd` uses the "GQvdG blocked with GQvdG unblocked" algorithm, with the minor modification that the algorithm switches to what is essentially our pure basic unblocked algorithm for the final 128×128 subproblem (when A_{BR} is 128×128).

Of particular interest is the comparison of the curves labeled "GQvdG blocked with GQvdG unblocked" and "blocked with lazy unblocked with fusing". Table 2 predicts that these should attain very similar performance. While the latter performs much better than its unfused counterpart, it does not attain the performance of "GQvdG blocked with GQvdG unblocked". One would expect it to be even more competitive if the fused operation were fully optimized.

6.6 Reduction to tridiagonal form

Figure 14 reports performance for various implementations of reduction to tridiagonal form. There is not much to remark upon here: the algorithms that use fused implementations do not perform well, possibly because the level-2 BLAS used by the basic algorithm are optimized to a degree that cannot be easily attained when implementing the fused the operations in terms of level-1 BLAS.

6.7 Reduction to bidiagonal form

Figure 15 reports performance for various implementations of reduction to bidiagonal form. For this operation there is a clear advantage gained from rearranging the computations and fusing operations. The "blocked with lazy rearranged with fusing" implementation closely tracks the performance of Howell's implementation and thus confirms the benefit of fusing. Howell's implementation implements the fused operation in terms of level-2 BLAS operations with a few columns while our implementation uses calls to level-1 BLAS operations, which accounts for the slightly better performance attained by his implementation.

6.8 Hybrid algorithms

In Figure 15 it can be observed that, for the smallest problem sizes ($n \leq 100$), the "basic unblocked with fusing" algorithm yields the best performance. Similarly, for a range of medium-sized problem sizes ($100 < n \leq 500$), the performance of the "rearranged unblocked with fusing" algorithm is superior. This suggests that a library routine should switch algorithms as a function of problem size. In Figure 16 we show performance for one such hybrid algorithm implementation.[5] There, "blocked with lazy rearranged with fusing (optimized)" refers to an implementation that uses the "basic unblocked with fusing" algorithm if the problem size is 100×100 or smaller, and then uses the "blocked with lazy rearranged with fusing" algorithm, except that it switches to the "rearranged unblocked with fusing" algorithm when the size of A_{BR} is less than 500×500. For a range of problem sizes this approach yields a slight advantage of up to 12 percent over netlib `dgebrd`.

Similar hybrid algorithms can constructed in a straightforward manner for both reduction to upper Hessenberg form and reduction to tridiagonal form, and so we have omitted results corresponding to those operations.

6.9 Experiments with multiple cores

A logical criticism of the experimental results given in the paper is that they only involve a single core. However, the limiting factor for performance is the bandwidth to memory which is clearly demonstrated by

[5]Note that the netlib LAPACK implementations of all three condensed form operations tested in this paper employ hybrid approaches, albeit with different crossover points. The netlib routines for reduction to upper Hessenberg form (`dgehrd`) and reduction to bidiagonal form (`dgebrd`) switch to basic unblocked algorithms for the final 128×128 submatrix, while the routine for reduction to tridiagonal form (`dsytrd`) switches for the final 32×32 submatrix.

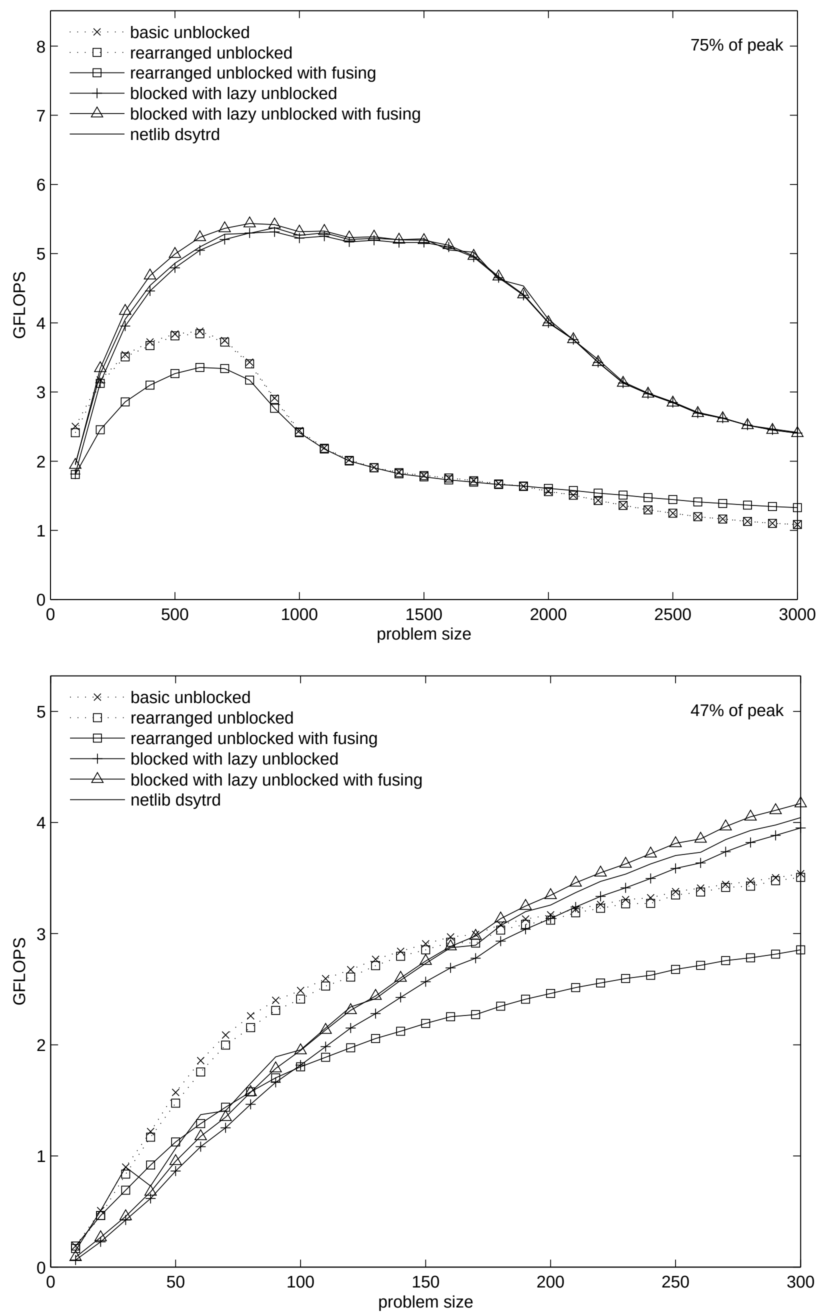

Figure 14: Performance of various implementations of reduction to tridiagonal form for problem sizes up to 3000 (top) and up to 300 (bottom). Implementations of blocked algorithms use a block size of 32.

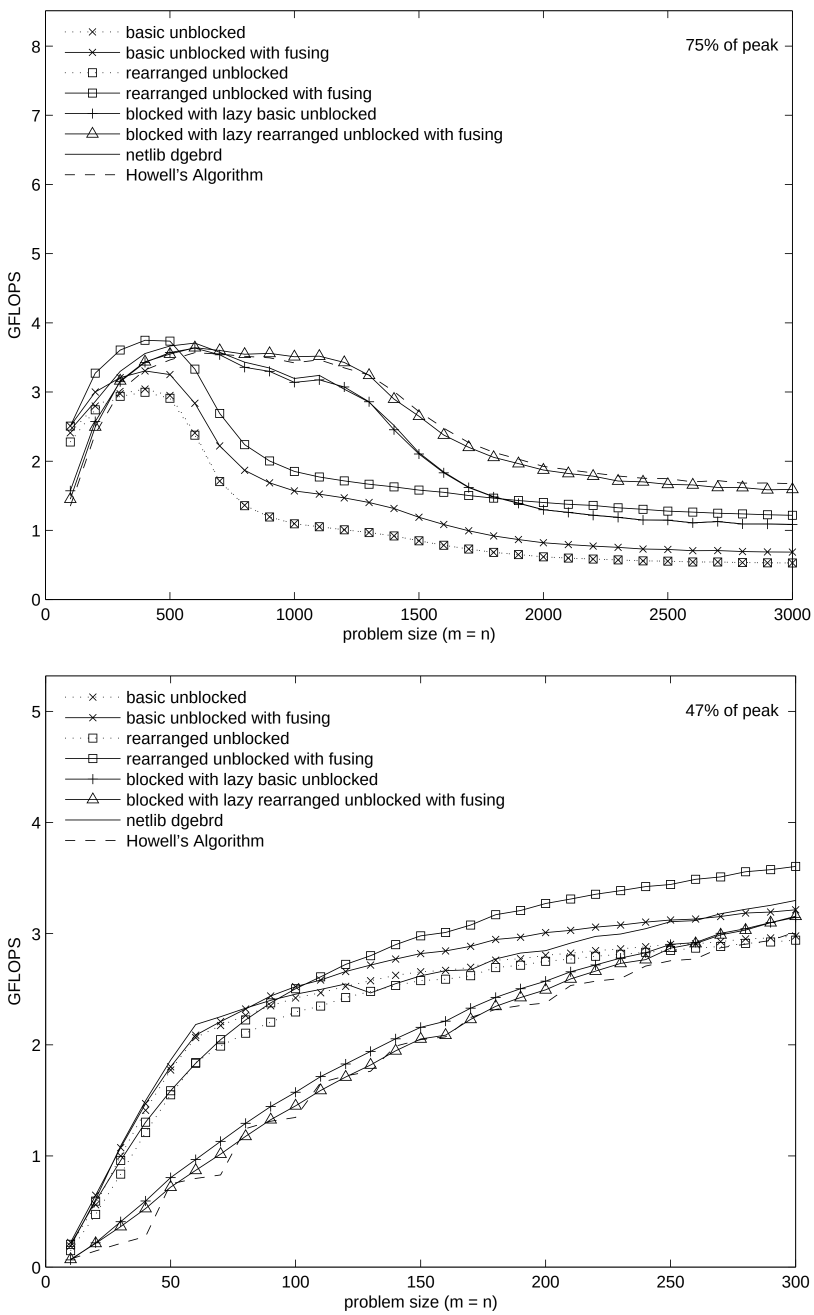

Figure 15: Performance of various implementations of reduction to bidiagonal form for problem sizes up to 3000 (top) and up to 300 (bottom). Implementations of blocked algorithms use a block size of 32.

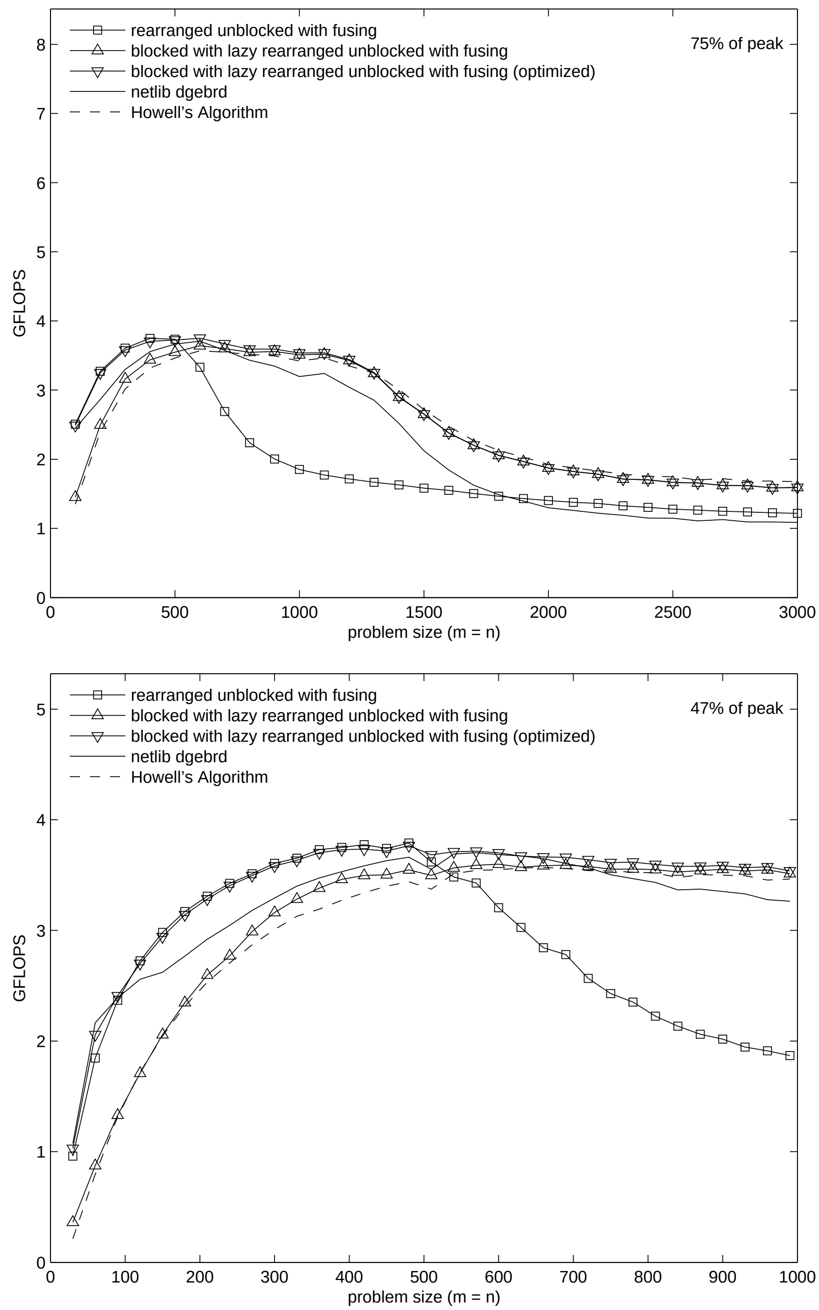

Figure 16: Performance of optimized blocked implementations of reduction to bidiagonal form for problem sizes up to 3000 (top) and up to 1000 (bottom). Implementations of blocked algorithms use a block size of 32.

the experiments. Also, parallelizing the fused operations goes beyond the scope of this paper. The work presented here exposes how algorithms can be rearranged to create fusable operations so that others can focus on the optimization of those operations.

7 Conclusion

This paper presents what we believe to be the most complete analysis to date of algorithms for reducing matrices to condensed form. Numerous algorithms are summarized and opportunities for rearranging and fusing of operations are exposed. For different ranges of problem sizes different algorithms attain the best performance.

At the time of this writing, our research group does not have the in-house expertise to fully optimize the fused operations. As a result, the performance results should be taken with a grain of salt. Conclusive evidence would come only when the fused operations are assembly-coded. The implementations and related timing experiments are part of the `libflame` library so that others can push the envelop on performance even further.

Acknowledgments This research was partially sponsored by NSF grants OCI-0850750 and NSF CCF-0917167, and a grant from Microsoft.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide (third ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[2] 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 Trans. Math. Soft.*, 31(1):1–26, March 2005.

[3] Paolo Bientinesi, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application programming interfaces. *ACM Trans. Math. Soft.*, 31(1):27–59, March 2005.

[4] Christian Bischof and Charles Van Loan. The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput.*, 8(1):s2–s13, Jan. 1987.

[5] Basic linear algebra subprograms technical forum standard. *International Journal of High Performance Applications and Supercomputing*, 16(1), Spring 2002.

[6] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[7] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[8] Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, PA, 1991.

[9] Jack J. Dongarra, Sven J. Hammarling, and Danny C. Sorensen. Block reduction of matrices to condensed forms for eigenvalue computations. *Journal of Computational and Applied Mathematics*, 27, 1989.

[10] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.

[11] Gary Howell. Fortran 77 codes for Householder bidiagonalization. `http://www.ncsu.edu/itd/hpc/Documents/Publications/gary_howell/030905.tar`, 2005.

[12] Gary W. Howell, James W. Demmel, Charles T. Fulton, Sven Hammarling, and Karen Marmol. Cache efficient bidiagonalization using BLAS 2.5 operators. *ACM Transactions on Mathematical Software*, 34(3):14:1–14:33, May 2008.

[13] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Ortí, Robert van de Geijn, and Field Van Zee. Accumulating Householder transformations, revisited. *ACM Transactions on Mathematical Software*, 32(2):169–179, June 2006.

[14] Representation of orthogonal or unitary matrices. `http://www.netlib.org/lapack/lug/node128.html`.

[15] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.

[16] R. B. Lehoucq. LAPACK Working Note 72: The computation of elementary unitary matrices. Technical Report CS-94-233, University of Tennessee, 1994.

[17] C. Puglisi. Modification of the Householder method based on the compact wy representation. *SIAM J. Sci. Stat. Comput.*, 13:723–726, 1992.

[18] Enrique S. Quintana, Gregorio Quintana, Xiaobai Sun, and Robert van de Geijn. A note on parallel matrix inversion. *SIAM J. Sci. Comput.*, 22(5):1762–1771, 2001.

[19] Gregorio Quintana-Ortí and Robert van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Transactions on Mathematical Software*, 32(2):180–194, June 2006.

[20] Robert Schreiber and Charles Van Loan. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, Jan. 1989.

[21] Jeremy G. Siek, Ian Karlin, and Elizabeth R. Jessup. Build to order linear algebra kernels. April 2009.

[22] Xiaobai Sun. Aggregations of elementary transformations. Technical Report Technical report DUKE–TR–1996–03, Duke University, 1996.

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

[24] Field G. Van Zee. *libflame: The Complete Reference*. `www.lulu.com`, 2010.

[25] Field G. Van Zee, Ernie Chan, Robert A. van de Geijn, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. The libflame library for dense matrix computations. *Computing in Science and Engineering*, 11:56–63, 2009.

[26] Richard M. Veras, Jonathan S. Monette, Field G. Van Zee, Robert A. van de Geijn, and Enrique S. Quintana-Ortí. FLAMES2S: From abstraction to high performance. *ACM Trans. Math. Soft.* submitted. Available from `http://z.cs.utexas.edu/wiki/flame.wiki/Publications/`.

[27] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9(1):152–163, 1988.

A Computing in the complex domain

For simplicity and clarity, the algorithms given thus far have assumed computation on real matrices. In this appendix, we briefly discuss how to formulate a few of these algorithms for complex matrices.

In order to capture more generalized algorithms which work in both the real and complex domains, we must first introduce a complex Householder transform.

Definition 2 *Let $u \in \mathbb{C}^n$, $\tau \in \mathbb{R}$. Then $H = H(u) = I - \tau^{-1} u u^H$, where $\tau = \frac{1}{2} u^H u$, is a complex Householder transformation.*

The complex Householder transform has properties similar to those of the real instantiation, namely: (1) $HH = I$; (2) $H = H^H$, and so $H^H H = H H^H = I$; and (3) if $H_0, \cdots, H_{k-1}$ are complex Householder transformations and $Q = H_0 H_1 \cdots H_{k-1}$, then $Q^H Q = Q Q^H = I$.

Let $x, v, u \in \mathbb{C}^n$,

$$x \rightarrow \left(\frac{\chi_1}{x_2} \right), v \rightarrow \left(\frac{\nu_1}{v_2} \right), u \rightarrow \left(\frac{\upsilon_1}{u_2} \right),$$

$v = x - \alpha e_0$, and $u = v/\nu_1$. We can re-express the complex Householder transform H as:

$$H = \left(I - \tau^{-1} \left(\frac{1}{u_2} \right) \left(\frac{1}{u_2} \right)^H \right)$$

It can be shown that the application of $H(u)$ to a vector x,

$$H \left(\frac{\chi_1}{x_2} \right) = \left(\frac{\alpha}{0} \right) \tag{4}$$

is satisfied for

$$\alpha = -\frac{\|x\|_2 \chi_1}{|\chi_1|}.$$

Notice that for $x, v, u \in \mathbb{R}^n$, this definition of α is equivalent to the definition given for real Householder transformations in Section 2.2, since $\chi_1/|\chi_1| = \text{sign}(\chi_1)$. By re-defining α this way, we allow τ to remain real, which allows the complex Householder transform to retain the property of being a reflector. Other instances of the Householder transform, such as those found in LAPACK, restrict α to the real domain [14, 16]. In these situations, Eq. (4) is only satisfiable if $\tau \in \mathbb{C}$, which results in $HH \neq I$. We prefer our Householder transforms to remain reflectors in both the real and complex domains, and so we choose to define α as above.

Recall that Figures 1–12 illustrate algorithms for computing on real matrices. We will now review a few of the algorithms, as expressed in terms of the complex Householder transform.

A.1 Reduction to upper Hessenberg form

Since the complex Householder transform H is a reflector, the basic unblocked algorithm for reducing a complex matrix to upper Hessenberg is, at a high level, identical to the algorithm for real matrices:

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right)$.
- Let $[u_{21}, \tau, a_{21}] := \textsc{Housev}(a_{21})$.
- Update

$$\left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) := \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & H \end{array} \right) \left(\begin{array}{c|c} a_{01} & A_{02} \\ \hline \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array} \right) \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & H \end{array} \right) = \left(\begin{array}{c|c} a_{01} & A_{02} H \\ \hline \alpha_{11} & a_{12}^T H \\ \hline H a_{21} & H A_{22} H \end{array} \right)$$

  where $H = H(u_{21})$. Note that $a_{21} := H a_{21}$ need not be executed since this update was performed by the instance of $\textsc{Housev}$ above.

Algorithm: $[A] := \text{COMPLEXHESSRED\_UNB}(b, A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $u \rightarrow \left(\begin{array}{c} u_T \\ \hline u_B \end{array}\right)$, $y \rightarrow \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right)$, $z \rightarrow \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right)$

where A_{TL} is 0×0 and u_T, y_T, and z_T have 0 rows

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} u_T \\ \hline u_B \end{array}\right) \rightarrow \left(\begin{array}{c} u_{01} \\ \hline \upsilon_{11} \\ \hline u_{21} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \rightarrow \left(\begin{array}{c} y_{01} \\ \hline \psi_{11} \\ \hline y_{21} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \rightarrow \left(\begin{array}{c} z_{01} \\ \hline \zeta_{11} \\ \hline z_{21} \end{array}\right)$$

where α_{11}, υ_{11}, ψ_{11}, ζ_{11} are scalars

Basic unblocked 1:

$[u_{21}, \tau, a_{21}] := \text{HOUSEV}(a_{21})$

$A_{22} := (I - u_{21}u_{21}^H/\tau)A_{22} = A_{22} - u_{21}u_{21}^H A_{22}/\tau$

$$\left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} A_{02} \\ \hline a_{12}^T \\ \hline A_{22} \end{array}\right)(I - u_{21}u_{21}^H/\tau) = \left(\begin{array}{c} A_{02} - A_{02}u_{21}u_{21}^H/\tau \\ \hline a_{12}^T - a_{12}^T u_{21}u_{21}^H/\tau \\ \hline A_{22} - A_{22}u_{21}u_{21}^H/\tau \end{array}\right)$$

Basic unblocked 2:	Rearranged unblocked:	
	$\alpha_{11} := \alpha_{11} - \upsilon_1\bar{\psi}_1 - \zeta_1\bar{\upsilon}_1$	$(\star)$
	$a_{12}^T := a_{12}^T - \upsilon_1 y_{21}^H - \zeta_1 u_{21}^H$	$(\star)$
	$a_{21} := a_{21} - u_{21}\bar{\psi}_1 - z_{21}\bar{\upsilon}_1$	$(\star)$
$[u_{21}, \tau, a_{21}] := \text{HOUSEV}(a_{21})$	$[x_{21}, \tau, a_{21}] := \text{HOUSEV}(a_{21})$	
	$A_{22} := A_{22} - u_{21}y_{21}^H - z_{21}u_{21}^H$	$(\star)$
$y_{21} := A_{22}^H u_{21}$	$v_{21} := A_{22}^H x_{21}$	
$z_{21} := A_{22}u_{21}$	$w_{21} := A_{22}x_{21}$	
	$u_{21} := x_{21}$; $y_{21} := v_{21}$	
	$z_{21} := w_{21}$	
$\beta := u_{21}^H z_{21}/2$	$\beta := u_{21}^H z_{21}/2$	
$y_{21} := (y_{21} - \bar{\beta}u_{21}/\tau)/\tau$	$y_{21} := (y_{21} - \bar{\beta}u_{21}/\tau)/\tau$	
$z_{21} := (z_{21} - \beta u_{21}/\tau)/\tau$	$z_{21} := (z_{21} - \beta u_{21}/\tau)/\tau$	
$A_{22} := A_{22} - u_{21}y_{21}^H - z_{21}u_{21}^H$		
$a_{12}^T := a_{12}^T - a_{12}^T u_{21}u_{21}^H/\tau$	$a_{12}^T := a_{12}^T - a_{12}^T u_{21}u_{21}^H/\tau$	
$A_{02} := A_{02} - A_{02}u_{21}u_{21}^H/\tau$	$A_{02} := A_{02} - A_{02}u_{21}u_{21}^H/\tau$	

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right),$$

$$\left(\begin{array}{c} u_T \\ \hline u_B \end{array}\right) \leftarrow \left(\begin{array}{c} u_{01} \\ \hline \upsilon_{11} \\ \hline u_{21} \end{array}\right), \left(\begin{array}{c} y_T \\ \hline y_B \end{array}\right) \leftarrow \left(\begin{array}{c} y_{01} \\ \hline \psi_{11} \\ \hline y_{21} \end{array}\right), \left(\begin{array}{c} z_T \\ \hline z_B \end{array}\right) \leftarrow \left(\begin{array}{c} z_{01} \\ \hline \zeta_{11} \\ \hline z_{21} \end{array}\right)$$

endwhile

Figure 17: Unblocked reduction to upper Hessenberg form using a complex Householder transform. Left: basic algorithm. Right: rearranged algorithm so that operations can be fused. Operations marked with $(\star)$ are not executed during the first iteration.

- Continue this process with the updated A_{22}.

As before, Ha_{21} is computed by HOUSEV.
The real and complex algorithms begin to differ with the updates of a_{12}^T and A_{02}:

$$\begin{aligned} a_{12}^T &:= a_{12}^T H \\ &= a_{12}^T - a_{12}^T u_{21} u_{21}^H/\tau \\ A_{02} &:= A_{02} H \\ &= A_{02} - A_{02} u_{21} u_{21}^H/\tau \end{aligned}$$

Specifically, we can see that u_{21} is conjugate-transposed instead of simply transposed.
The remaining differences can be seen by inspecting the update of A_{22}:

$$\begin{aligned} A_{22} &:= HA_{22}H \\ &= (I - u_{21}u_{21}^H/\tau)A_{22}(I - u_{21}u_{21}^H/\tau) \\ &= A_{22} - u_{21}(\underbrace{A_{22}^H u_{21}}_{v_{21}})^H/\tau - (\underbrace{A_{22}u_{21}}_{w_{21}})u_{21}^H/\tau + (u_{21}^H \underbrace{A_{22}u_{21}}_{w_{21}})u_{21}u_{21}^H/\tau^2 \\ &= A_{22} - u_{21}v_{21}^H/\tau - w_{21}u_{21}^H/\tau + \underbrace{u_{21}^H w_{21}}_{2\beta} u_{21}u_{21}^H/\tau^2 \\ &= A_{22} - u_{21}(v_{21}^H - \beta u_{21}^H/\tau)/\tau - ((w_{21} - \beta u_{21}/\tau)/\tau)u_{21}^H \\ &= A_{22} - u_{21} \underbrace{((v_{21} - \bar{\beta} u_{21}/\tau)/\tau)^H}_{y_{21}} - \underbrace{((w_{21} - \beta u_{21}/\tau)/\tau)}_{z_{21}} u_{21}^H \\ &= A_{22} - (u_{21}y_{21}^H + z_{21}u_{21}^H) \end{aligned}$$

This leads towards the basic and rearranged unblocked algorithms in Figure 17.

A.2 Reduction to tridiagonal form

Let $A \in \mathbb{C}^{n\times n}$ be Hermitian. If $A \rightarrow QBQ^H$ where B is upper Hessenberg and Q is unitary, then B is Hermitian and therefore tridiagonal. We may take advantage of the Hermitian structure of A just as we did with symmetry in Section 4. Let us assume that only the lower triangular part of A is stored and read, and that only the lower triangular part is overwritten by B.

When matrix A is Hermitian, and only the lower triangular part is referenced, the unblocked algorithms for reducing A to upper Hessenberg form can be changed by noting that $v_{21} = w_{21}^H$ and $y_{21} = z_{21}^H$. This results in the basic and rearranged unblocked algorithms shown in Figure 18.

A.3 Reduction to bidiagonal form

The basic algorithm for reducing a complex matrix to bidiagonal form can be explained as follows:

- Partition $A \rightarrow \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right)$.
- Let $\left[\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array}\right)\right] := \text{HOUSEV}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)\right)$.
- Update

$$\begin{aligned} \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) &:= \left(I - \left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right)^H /\tau_L\right)\left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right) \\ &= \left(\begin{array}{c|c} \alpha - \psi_{11}/\tau_L & a_{12}^T - y_{21}^T/\tau_L \\ \hline 0 & A_{22} - u_{21}y_{21}^T/\tau_L \end{array}\right), \end{aligned}$$

Algorithm: $[A] := \textsc{ComplexTriRed\_unb}(b, A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$, $x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array} \right)$

for $x \in \{u, y\}$

where A_{TL} is 0×0 and u_T, y_T have 0 rows

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array} \right) \rightarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array} \right)$$

for $(x, \chi) \in \{(u, \upsilon), (y, \psi)\}$

where α_{11}, υ_{11}, and ψ_{11} are scalars

Basic unblocked:	Rearranged unblocked:	
	$\alpha_{11} := \alpha_{11} - \upsilon_{11}\bar{\psi}_{11} - \psi_{11}\bar{\upsilon}_{11}$	$(\star)$
	$a_{21} := a_{21} - (u_{21}\bar{\psi}_{11} + y_{21}\bar{\upsilon}_{11})$	$(\star)$
$[u_{21}, \tau, a_{21}] := \textsc{Housev}(a_{21})$	$[x_{21}, \tau, a_{21}] := \textsc{Housev}(a_{21})$	
	$A_{22} := A_{22} - u_{21}y_{21}^H - y_{21}u_{21}^H$	$(\star)$
$y_{21} := A_{22}u_{21}$	$v_{21} := A_{22}x_{21}$	
	$u_{21} := x_{21}$; $y_{21} := v_{21}$	
$\beta := u_{21}^H y_{21}/2$	$\beta := u_{21}^H y_{21}/2$	
$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$	$y_{21} := (y_{21} - \beta u_{21}/\tau)/\tau$	
$A_{22} := A_{22} - u_{21}y_{21}^H - y_{21}u_{21}^H$		

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array} \right) \leftarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array} \right)$$

for $(x, \chi) \in \{(u, \upsilon), (y, \psi)\}$

endwhile

Figure 18: Unblocked reduction to tridiagonal form using a complex Householder transformation. Left: basic algorithm. Right: rearranged to allow fusing of operations. Operations marked with $(\star)$ are not executed during the first iteration.

where $\psi_{11} = \alpha_{11} + u_{21}^H a_{21}$ and $y_{21}^T = a_{12}^T + u_{21}^H A_{22}$. Note that $\alpha_{11} := \alpha - \psi_{11}/\tau_L$ need not be executed since this update was performed by the instance of Housev above.

- Let $[v_{21}, \tau_R, a_{12}] := \textsc{Housev}\,(a_{12})$.
- Update $A_{22} := A_{22}(I - v_{21}v_{21}^T/\tau_R) = A_{22} - z_{21}v_{21}^T/\tau_R$, where $z_{21} = A_{22}v_{21}$.
- Continue this process with the so updated A_{22}.

The resulting unblocked algorithm and a rearranged variant that allows fusing are given in Figure 19.

A.4 Blocked algorithms

Blocked algorithms may be constructed for reduction to upper Hessenberg form by making the following minor changes to the algorithms shown in Figure 5:

- For Algorithms 1–4, update A_{TR} by applying the complex block Householder transform, $(I - U_B T^{-1} U_B^H)$, instead of $(I - U_B T^{-1} U_B^T)$.
- For Algorithm 3, update A_{22} as $A_{22} = A_{22} - U_2 Y_2^H - Z_2 U_2^H$.

Algorithm: $[A] := \textsc{ComplexBiRed\_unb}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $x \rightarrow \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right)$

for $x \in \{u, v, y, z\}$

where A_{TL} is 0×0, u_T, v_T, y_T, z_T have 0 elements

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \rightarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array}\right)$$

for $(x, \chi) \in \{(u, \upsilon), (v, \nu), (y, \psi), (z, \zeta)\}$

where α_{11}, υ_{11}, ν_{11}, ψ_{11}, and ζ_{11} are scalars

Basic unblocked:

$$\left[\left(\begin{array}{c} 1 \\ \hline u_{21} \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array}\right)\right] := \textsc{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)\right)$$

$y_{21} := a_{12} + A_{22}^T \bar{u}_{21}$

$a_{12}^T := a_{12}^T - y_{21}^T / \tau_L$

$[v_{21}, \tau_R, a_{12}] := \textsc{Housev}(a_{12})$

$\beta := y_{21}^T \bar{v}_{21}$

$y_{21} := y_{21} / \tau_L$

$z_{21} := (A_{22} \bar{v}_{21} - \beta u_{21} / \tau_L) / \tau_R$

$A_{22} := A_{22} - u_{21} y_{21}^T - z_{21} v_{21}^T$

Rearranged unblocked:

$\alpha_{11} := \alpha_{11} - \upsilon_{11} \psi_{11} - \zeta_{11} \nu_{11}$ $(\star)$

$a_{21} := a_{21} - u_{21} \psi_{11} - z_{21} \nu_{11}$ $(\star)$

$a_{12}^T := a_{12}^T - \upsilon_{11} y_{21}^T - \zeta_{11} v_{21}^T$ $(\star)$

$$\left[\left(\begin{array}{c} 1 \\ \hline u_{21}^+ \end{array}\right), \tau_L, \left(\begin{array}{c} \alpha_{11} \\ \hline 0 \end{array}\right)\right] := \textsc{Housev}\left(\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)\right)$$

$a_{12}^+ := a_{12} - a_{12} / \tau_L$

$A_{22} := A_{22} - u_{21} y_{21}^T - z_{21} v_{21}^T$ $(\star)$

$y_{21} := A_{22}^T \bar{u}_{21}^+$

$a_{12}^+ := a_{12}^+ - y_{21} / \tau_L$

$w_{21} := A_{22} \bar{a}_{12}^+$

$y_{21} := y_{21} + a_{12}$

$[\psi_{11} - \alpha_{12}, \tau_R, \alpha_{12}] := \textsc{Houses}(a_{12}^+)$

$v_{21} := (a_{12}^+ - \alpha_{12} e_0) / (\psi_{11} - \alpha_{12})$;

$a_{12}^T := \alpha_{12} e_0^T$

$u_{21} := u_{21}^+$

$\beta := y_{21}^T \bar{v}_{21}$

$y_{21} := y_{21} / \tau_L$

$z_{21} := (w_{21} - \bar{\alpha}_{12} A_{22} e_0) / (\bar{\psi}_{11} - \bar{\alpha}_{12})$

$z_{21} := z_{21} - \beta u_{21} / \tau_L$

$z_{21} := z_{21} / \tau_R$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array}\right) \leftarrow \left(\begin{array}{c} x_{01} \\ \hline \chi_{11} \\ \hline x_{21} \end{array}\right)$$

for $(x, \chi) \in \{(u, \upsilon), (v, \nu), (y, \psi), (z, \zeta)\}$

endwhile

Figure 19: Unblocked reduction to bidiagonal form using a complex Householder transformation. Left: basic algorithm. Right: rearranged to allow fusing of operations. Operations marked with $(\star)$ are not executed during the first iteration.

- Compute T as $T = \frac{1}{2}D + S$ where $U_B^H U_B = S^H + D + S$.

Blocked algorithms for reduction to tridiagonal form and bidiagonal form can be constructed in a similar fashion.