SIAM J. Sci. Comput.
Vol. 39, No. 2, pp. C96–C115

HOUSEHOLDER QR FACTORIZATION WITH RANDOMIZATION FOR COLUMN PIVOTING (HQRRP)[*]

PER-GUNNAR MARTINSSON[†], GREGORIO QUINTANA ORTÍ[‡], NATHAN HEAVNER[†], AND ROBERT VAN DE GEIJN[§]

Abstract. A fundamental problem when adding column pivoting to the Householder QR factorization is that only about half of the computation can be cast in terms of high performing matrix-matrix multiplications, which greatly limits the benefits that can be derived from so-called blocking of algorithms. This paper describes a technique for selecting groups of pivot vectors by means of randomized projections. It is demonstrated that the asymptotic flop count for the proposed method is $2mn^2 - (2/3)n^3$ for an $m \times n$ matrix, identical to that of the best classical unblocked Householder QR factorization algorithm (with or without pivoting). Experiments demonstrate acceleration in speed of close to an order of magnitude relative to the GEQP3 function in LAPACK, when executed on a modern CPU with multiple cores. Further, experiments demonstrate that the quality of the randomized pivot selection strategy is roughly the same as that of classical column pivoting. The described algorithm is made available under open source license and can be used with LAPACK or `libflame`.

Key words. QR factorization, Householder QR, pivoting strategy, randomized algorithm, blocked algorithm

AMS subject classification. 65F05

DOI. 10.1137/16M1081270

1. Introduction.

The QR factorization is a staple of linear algebra, with applications ranging from linear least-squares solution of overdetermined systems to the identification of low rank approximation via the determination of an approximate orthonormal basis for the column space. Standard algorithms for computing the QR factorization include Gram–Schmidt orthogonalization and those based on Householder transformations (reflectors). When it is desirable for the QR factorization to also reveal the approximate rank of the original matrix, it is important that the elements of the diagonal of R be ordered with larger elements in magnitude appearing earlier. In this case, column pivoting (swapping) is employed during the QR factorization, yielding QR factorization with column pivoting (QRP). It is well known that the Householder QR factorization (HQR) yields columns of Q that are orthogonal to a high degree of precision, making these algorithms the weapon of choice in many situations. Pivoting can be added to HQR to yield HQR with column pivoting (HQRP). This topic is covered by standard texts on numerical linear algebra [13].

To achieve high performance for dense linear algebra algorithms, so-called blocked algorithms are employed that cast most computation in terms of matrix-matrix operations supported by the widely used level-3 Basic Linear Algebra Subprograms (BLAS)

[*] Submitted to the journal's Software and High-Performance Computing section June 23, 2016; accepted for publication (in revised form) November 8, 2016; published electronically April 11, 2017.
http://www.siam.org/journals/sisc/39-2/M108127.html

Funding: This work was supported by DARPA under contract N66001-13-1-4050 and by the NSF under contracts DMS-1407340, DMS-1620472, and ACI-1148125/1340293.

[†] Department of Applied Mathematics, University of Colorado at Boulder, Boulder, CO 80309-0526 (martinss@colorado.edu, nathan.heavner@colorado.edu).

[‡] Departmento de Ingeniería y Ciencia de Computadores, Universitat Jaume I, 12.071 Castellón, Spain (gquintan@icc.uji.es).

[§] Department of Computer Science and Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX 78712 (rvdg@cs.utexas.edu).

[7, 8] because such operations can be implemented to achieve very high performance on modern processors via a combination of careful reuse of data in the caches and low level implementation in terms of assembly code or intrinsic vector operations. Widely used current implementations of the level-3 BLAS are based on techniques exposed by Goto and van de Geijn [15, 14] and available in open source libraries including OpenBLAS [39] (a fork of GotoBLAS) and BLIS [36], as well as vendor implementions including AMD's ACML [2], Intel's MKL [20], and IBM's ESSL [19] libraries.

The fundamental problem with the classical approach to HQRP is that only half of the computation can be cast in terms of GEMM, as described in the paper [31] that underlies LAPACK's `geqp3` routine [3]. This means that blocking can only improve performance by, at best, a factor of two, which is inherent from the fact that it must be known how remaining columns will be updated in order to compute the 2-norms of remaining columns. Bischof and Quintana-Ortí describe in a pair of papers [6, 5] an attempt to overcome this problem by using so-called window pivoting in combination with HQR. While much faster than `geqp3`, this approach is more complicated than the method proposed in this paper and never made it into LAPACK.

The present paper proposes to solve the problem by means of randomized projections. To describe the idea, suppose that we seek to determine a set of b good pivot columns in an $m \times n$ matrix A. We then draw a Gaussian random matrix G of size $b \times m$ and form a $b \times n$ *sampling matrix* $Y = GA$. Once Y is available, we execute QRP on this matrix to find the b pivot columns. This computation is efficient since Y is small compared to A (it has only b rows) and results in good pivot choices since the random projection produces a matrix Y that has approximately the same linear dependencies between its columns as does A.[1] With this observation, it becomes easy to block the HQR factorization with column pivoting. At each iteration of the blocked algorithm, we use the randomized sampling approach to identify a set of b columns that are then moved to the front of the actual matrix, at which point a regular step of HQR can be used to move the computation forward, optionally with additional column pivoting only within a narrow panel of the matrix. Importantly, the sampling matrix can be cheaply downdated rather than recomputed at each step, allowing the performance of the proposed algorithm to asymptotically approach that of a standard blocked HQR implementation that does not pivot columns. Figure 1 illustrates the dramatic performance improvements that are realized.

The idea to use randomized sampling to pick blocks of pivot vectors was first published by Martinsson in May 2015 [25]. A very similar technique was published by Duersch and Gu in September 2015 [10]. The observation that downdating of the sampling matrix enables the randomized scheme to attain the same asymptotic flop count as classical HQRP was discovered independently by the two groups and was first published in [10]. More broadly, the idea that one can select a subset of columns of a matrix that forms a good approximate basis for the column space of the matrix by performing QRP on a small matrix whose rows are random linear combinations of the rows of the original matrix was first described in [26, sect. 4.1] and later elaborated in [23, 18, 27]. This problem is closely related to the problem of finding a set of columns of maximal spanning volume [16] and to the problem of finding so-called CUR and interpolative decompositions [37]. These ideas tie in to a larger literature on randomized techniques for computing low-rank approximations of matrices that includes [12, 9, 24, 28].

[1] To be precise, for linear dependencies to be preserved reliably, one needs to perform a very slight amount of oversampling. See section 3.1 for details.

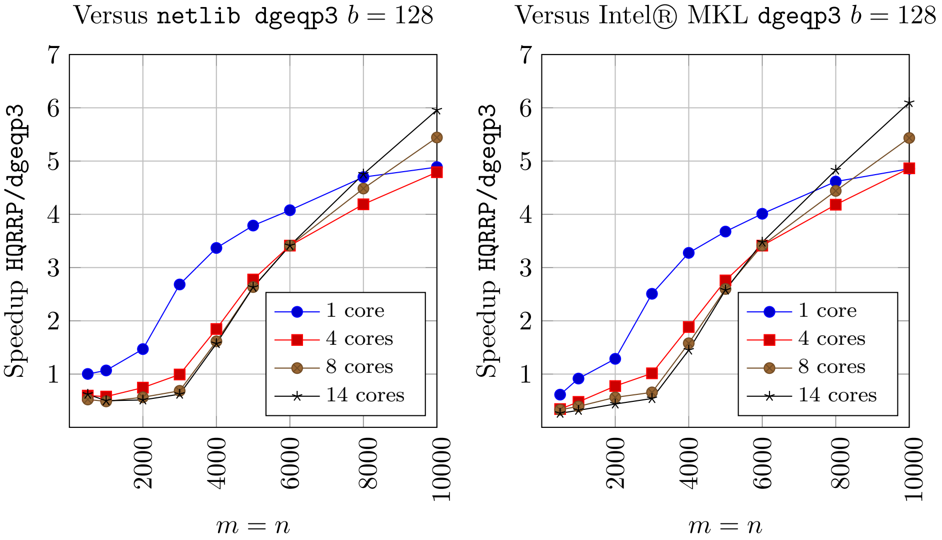

FIG. 1. *Speedup of new blocked HQR factorization with randomized column pivoting (*HQRRP*) relative to LAPACK's faster routine (*dgeqp3*) on a* 14*-core Intel Xeon* E5-2695 *v*3*; see section* 4.1 *for details.*

This paper describes a practical implementation of the proposed method that can be incorporated in libraries like LAPACK and libflame [34, 35]. Implementation details that are important for attaining high practical performance are described to enable readers to reproduce and extend the ideas. The paper provides a cost analysis that shows that asymptotically the number of floating point operations approaches that of HQR without pivoting while most computation is cast in terms of matrix-matrix multiplication like the corresponding blocked HQR without pivoting. It reports unprecedented performance for pivoted QR factorization on current architectures and provides empirical quality results. Importantly, the implementation is made available for use by the computational science community under an open source license. The conclusion discusses how these results pave the way for future opportunities.

The paper is organized as follows. Section 2 lists some standard facts about Householder reflectors and pivoted QR factorizations that we need in the presentation. Section 3 describes how the classical HQR factorization algorithm can be blocked by using randomization in the pivot selection step. Section 4 reports the results from numerical experiments investigating the speed of the algorithm and the quality of the pivoting selection strategy. Sections 5 summarizes the key results and discusses future work. Section 6 describes publicly available software that implements the techniques presented.

2. Householder QR factorization.

In this section, we briefly review the state of the art regarding Householder factorization based on Householder transformations (HQR). Throughout, we use the FLAME notation for representing dense linear algebra algorithms [30, 17]. In particular, for any matrix X, we let $m(X)$ and $n(X)$ denote the number of rows and columns of X, respectively.

2.1. Householder transformations (reflectors).

A standard topic in numerical linear algebra is the concept of a reflector, also known as a Householder transformation [13]. The review in this subsection follows [21], in which a similar notation is also employed.

Given a nonzero vector $u \in \mathbb{C}^n$, the matrix $H(u) = I - \frac{1}{\tau} u u^H$ with $\tau = \frac{u^H u}{2}$ has the property that it reflects a vector to which it is applied with respect to the subspace orthogonal to u. Given a vector x, the vector u and scalar τ can be chosen so that $H(u)x$ equals a multiple of e_0, the first column of the identiy matrix. Furthermore, u can be normalized so that its first element equals one.

In our discussions, given a vector $x = \left(\frac{\chi_1}{x_2}\right)$, the function $[\left(\frac{\rho}{u_2}\right), \tau] := \text{Housev}(\left(\frac{\chi_1}{x_2}\right))$ computes the vector $u = \left(\frac{1}{u_2}\right)$ and $\tau = \frac{u^H u}{2}$ so that $H(u)x = \rho e_0$,

2.2. Unblocked Householder QR factorization. A standard unblocked algorithm for HQR of a given matrix $A \in \mathbb{C}^{m\times n}$, typeset using the FLAME notation, is given in Figure 2 (left). The body of the loop computes

$$\left[\left(\frac{\rho_{11}}{u_{21}}\right), \tau_{11}\right] := \text{Housev}\left(\left(\frac{\alpha_{11}}{a_{21}}\right)\right),$$

which overwrites a_{11} with ρ_{11} and a_{21} with u_{21}, after which the remainder of A is updated by

$$\left(\frac{a_{12}^T}{A_{22}}\right) := \left(I - \frac{1}{\tau_{11}}\left(\frac{1}{u_{21}}\right)\left(\frac{1}{u_{21}}\right)^H\right)\left(\frac{a_{12}^T}{A_{22}}\right).$$

Upon completion, the (Householder) vectors that define the Householder transformations have overwritten the elements in which they introduced zeroes, and the upper triangular part of A contains R. How the matrix T fits into the picture will become clear next.

2.3. The UT transform: Accumulating Householder transformations. Given $A \in \mathbb{C}^{n\times b}$, let U contain the Householder vectors computed during the HQR of A. Let us assume that $H(u_{b-1}) \cdots H(u_1)H(u_0)A = R$. Then there exists an upper triangular matrix so that $I - UT^{-H}U^H = H(u_{b-1}) \cdots H(u_1)H(u_0)$. The desired matrix T equals the strictly upper triangular part of $U^H U$ with the diagonal elements equal to $\tau_0, \ldots, \tau_{b-1}$. The matrix T can be computed during the unblocked HQR, as indicated in Figure 2 (left). In [21], the transformation $I - UT^{-1}U^H$ that equals the accumulated Householder transformations is called the *UT transform*. The UT transform is conceptually related to the more familiar WY transform [4] and compact WY transform [32]; see [21] for details on how the different representations relate to one another.

2.4. A blocked QR Householder factorization algorithm. A blocked algorithm for HQR that exploits the insights that resulted in the UT transform can now be described as follows. Partition

$$A \rightarrow \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right),$$

where A_{11} is $b \times b$. We can use the unblocked algorithm in Figure 2 (left) to factor the panel $\left(\frac{A_{11}}{A_{21}}\right)$, creating matrix T_{11} as a side effect. Now we need to also apply the UT transform to the rest of the columns:

```
Algorithm: $[A,T] := \text{HQR\_UNB}(A,T)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$,
  $T \rightarrow \left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right)$
  where $A_{TL}$ is $0 \times 0$, $T_{TL}$ is $0 \times 0$

while $m(A_{TL}) < m(A)$ do
  Repartition
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$,
  $\left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline 0 & \tau_{11} & t_{12}^T \\ \hline 0 & 0 & T_{22} \end{array}\right)$
    where $\alpha_{11}$ is $1 \times 1$, $\tau_{11}$ is $1 \times 1$

  $\left[\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right), \tau_{11}\right] := \text{Housev}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$
  $w_{12}^T := (a_{12}^T + a_{21}^H A_{22})/\tau_{11}$
  $\left(\begin{array}{c} a_{12}^T \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} a_{12}^T - w_{12}^T \\ \hline A_{22} - a_{21} w_{12}^T \end{array}\right)$
  $t_{01} := (a_{10}^T)^H + A_{20}^H a_{21}$

  Continue with
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$,
  $\left(\begin{array}{c|c} T_{TL} & T_{TR} \\ \hline 0 & T_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} T_{00} & t_{01} & T_{02} \\ \hline 0 & \tau_{11} & t_{12}^T \\ \hline 0 & 0 & T_{22} \end{array}\right)$
endwhile
```

```
Algorithm: $[A,T] := \text{HQR\_BLK}(A,T)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$,
  $T \rightarrow \left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right)$
  where $A_{TL}$ is $0 \times 0$, $T_T$ has 0 rows

while $m(A_{TL}) < m(A)$ do
  Determine block size $b$
  Repartition
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$,
  $\left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline T_2 \end{array}\right)$
    where $A_{11}$ is $b \times b$, $T_1$ has $b$ rows

  $[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), T_1] :=$
      $\text{HQR\_UNB}(\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), T_1)$
  $W_{12} := T_1^{-H}(U_{11}^H A_{12} + U_{21}^H A_{22})$
  $\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} A_{12} - U_{11} W_{12} \\ \hline A_{22} - U_{21} W_{12} \end{array}\right)$

  Continue with
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$,
  $\left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline T_2 \end{array}\right)$
endwhile
```

FIG. 2. *Left: Unblocked Householder transformation based QR factorization merged with the computation of T for the UT transform. Right: Blocked Householder transformation based QR factorization. In this algorithm, U_{11} is the unit lower triangular matrix stored below the diagonal of A_{11} and U_{21} is stored in A_{21}.*

$$\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(I - \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right) T_{11}^{-1} \left(\begin{array}{c} U_{11} \\ \hline U_{21} \end{array}\right)^H\right)^H \left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) = \left(\begin{array}{c} A_{12} - U_{11} W_{12} \\ \hline A_{22} - U_{21} W_{12} \end{array}\right),$$

where $W_{12} = T_{11}^{-H}(U_{11}^H A_{12} + U_{21}^H A_{22})$. This motivates the blocked HQR algorithm in Figure 2 (right), which we will refer to as HQR_BLK.

The benefit of the blocked algorithm is that it casts most computation in terms of the computations $U_{21}^H A_{22}$ (row panel times matrix multiply) and $A_{22} - U_{21} W_{12}$

(rank-b update). Such matrix-matrix multiplications can attain high performance by amortizing data movement between memory layers.

These insights form the basis for the LAPACK routine GEQRF (except that it uses a compact WY transform instead of the UT transform).

2.5. Householder QR factorization with column pivoting. An unblocked (rank-revealing) HQR factorization with column pivoting (HQRP) swaps the column of A_{BR} with largest 2-norm with the first column of that matrix at the top of the loop body. As a result, the diagonal elements of matrix R are ordered from largest to smallest in magnitude, which, for example, allows the resulting QR factorization to be used to identify a high-quality approximate low-rank orthonormal basis for the column space of A. (To be precise, column pivoted QR returns a high-quality basis in most cases but may produce strongly suboptimal results in rare situations. For details, see [22, 16], and the description of "Matrix 4" in section 4.2.)

The fundamental problem with the best known algorithm for HQRP, which underlies LAPACK's routine `geqp3`, is that it only casts half of the computation in terms of matrix-matrix multiplication [31]. The unblocked algorithm called from the blocked algorithm operates on the entire "remaining matrix" (A_{BR} in the blocked algorithm), computes b more Householder transforms and b more rows of R, computes the matrix W_2, and returns the information about how columns were swapped. In the blocked algorithm itself, only the update $A_{22} - A_{21}W_2$ remains to be performed. When only half the computation can be cast in terms of matrix-matrix multiplication, the resulting blocked algorithm is only about twice as fast as the unblocked algorithm.

3. Randomization to the rescue.

This section describes a computationally efficient technique for picking a selection of b columns from a given $n \times n$ matrix A that form good choices for the first b pivots in a blocked HQRP algorithm. Observe that this task is closely related to the problem of finding an index set s of length b such that the columns in $A(:, s)$ form a good approximate basis for the column space of A. Another way of expressing this problem is that we are looking for a collection of b columns whose spanning volume in $\mathbb{C}^n$ is close to maximal. To find the absolutely optimal choice here is a hard problem [16], but luckily, for pivoting purposes it is sufficient to find a choice that is "good enough."

3.1. Randomized pivot selection. The strategy that we propose is simple. The idea is to perform classical QR factorization with column pivoting (QRP) on a matrix Y that is much smaller than A, so that performing QRP with that matrix constitutes a lower order cost. As a bonus, it may fit in fast cache memory. This smaller matrix can be constructed by forming random linear combinations of the rows of A as follows:

1. Fix an oversampling parameter p. Setting $p = 5$ or $p = 10$ are good choices.
2. Form a random matrix G of size $(b + p) \times n$ whose entries are drawn independently from a normalized Gaussian distribution.
3. Form the $(b + p) \times n$ sampling matrix $Y = GA$.

The sampling matrix Y has as many columns as A, but many fewer rows. Now execute b steps of a column pivoted QR factorization to determine an integer vector with b elements that capture how columns need to be pivoted:

$$s = \textsc{DeterminePivots}(Y, b).$$

In other words, the columns $Y(:, s)$ are good pivot columns for Y. Our claim is that due to the way Y is constructed, the columns $A(:,s)$ are then also good choices for pivot columns of A. This claim is supported by extensive numerical experiments, some of which are given in section 4.2. There is theory supporting the claim that these b columns form a good approximate basis for the column space of A (see, e.g., [18, sect. 5.2] and [23, 37]), but it has not been directly proven that they form good choices as pivots in a QR factorization. This should not be surprising given that it is *known* that even classical column pivoting can result in poor choices [22]. Known algorithms that are provably good are all far more complex to implement [16].

Notice that there are many choices of algorithms that can be employed to determine the pivots. For example, since high numerical accuracy is not necessary, the classical modified Gram–Schmidt algorithm with column pivoting is a simple yet effective choice.

The randomized strategy described here for determining a block of pivot vectors is inspired by a technique published in [26, sect. 4.1] for computing a low-rank approximation to a matrix and later elaborated in [23, 18, 27].

Remark 1 (choice of oversampling parameter p). The reliability of the procedures described in this section depends on the choice of the oversampling parameter p. It is well understood how large p needs to be in order to determine a high-quality approximate basis for the column space of A with extremely high reliability: the choice $p = 5$ is very good, $p = 10$ is excellent, and $p = b$ is almost always overkill [18]. The pivot selection problem is less well studied but is more forgiving. (The choice of pivots does not necessarily have to be particularly optimal.) Numerical experiments indicate that even setting $p = 0$ typically results in good choices. However, the choices $p = 5$ or $p = 10$ appear to be good generic values that have resulted in excellent choices in every experiment we have run.

Remark 2 (intuition of random projections). To understand why the pivot columns selected by processing the small matrix Y also form good choices for the original matrix A, it might be helpful to observe that for a Gaussian random matrix G of size $\ell \times n$, it is the case that for any $x \in \mathbb{R}^n$, we have $\mathbb{E}\big[\|Gx\|^2\big] = \|x\|^2$, where $\mathbb{E}$ denotes expectation. Moreover, as the number of rows ℓ grows, the probability distribution of $\|Gx\|$ concentrates tightly around its expected value; see, e.g., [38, sect. 2.4] and the references therin. This means that for any pair of indices $i, j \in \{1, 2, \ldots, n\}$ we have $\mathbb{E}\big[\|Y(:,i) - Y(:,j)\|^2\big] = \|A(:,i) - A(:,j)\|^2$. This simple observation does not in any way provide a proof that the randomized strategy we propose works, but it might help understand the underlying intuition.

3.2. Efficient downdating of the sampling matrix Y.

For the QRP factorization algorithm, it is well known that one does not need to recompute the column norms of the remainder matrix after each step. Instead, these can be cheaply downdated, as described, e.g., in [33, Chap. 5, sect. 2.1]. In terms of asymptotic flop counts, this observation makes the cost of pivoting become a lower order term, and consequently both unpivoted and pivoted HQR algorithms have the same leading order term $(4/3)n^3$ in their asymptotic flop[2] counts for $n \times n$ matrices. In this section, we describe an analogous technique for the randomized sampling strategy described in

[2] We use the standard convention of counting one multiply and one add as one flop, regardless of whether a complex or real operation is performed.

section 3.1. This downdating strategy was discovered by one of the authors; a closely related technique was discovered independently by Duersch and Gu [10] in 2015.

First observe that if the randomized sampling technique described in section 3.1 is used in the obvious fashion, then each step of the iteration requires the generation of a Gaussian random matrix G and a matrix-matrix multiply involving the remaining portion of A in the lower right corner to form the sampling matrix Y. The number of flops required by the matrix-matrix multiplications add up to an $O(n^3)$ term for $n \times n$ matrices. However, it turns out to be possible to avoid computing a sampling matrix Y from scratch at every step. The idea is that if we select the randomizing matrix G in a particular way in every step beyond the first, then the corresponding sampling matrix Y can inexpensively be computed by downdating the sampling matrix from the previous step.

To illustrate, suppose that we start with an $n \times n$ original matrix $A = A^{(0)}$. In the first blocked step, we draw a $(b+p) \times n$ randomizing matrix $G^{(1)}$ and form the $(b+p) \times n$ sampling matrix

$$Y^{(1)} = G^{(1)} A^{(0)}. \tag{3.1}$$

Using the information in $Y^{(1)}$, we identify the b pivot vectors and form the corresponding permutation matrix $P^{(1)}$. Then the matrix $Q^{(1)}$ representing the b Householder reflectors dictated by the b pivot columns is formed. Applying these transforms to the right and the left of $A^{(0)}$, we obtain the matrix

$$A^{(1)} = \left(Q^{(1)}\right)^* A^{(0)} P^{(1)}. \tag{3.2}$$

To select the pivots in the next step, we need to form a randomizing matrix $G^{(2)}$ and a sampling matrix $Y^{(2)}$ that are related through

$$Y^{(2)} = G^{(2)}\left(A^{(1)} - R^{(1)}\right), \tag{3.3}$$

where $R^{(1)}$ holds the top b rows of $A^{(1)}$ so that

$$A^{(1)} - R^{(1)} = \begin{pmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & A_{22}^{(1)} \end{pmatrix} - \begin{pmatrix} A_{11}^{(1)} & A_{12}^{(1)} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & A_{22}^{(1)} \end{pmatrix}.$$

The key idea is now to *choose* the randomizing matrix $G^{(2)}$ according to the formula

$$G^{(2)} = G^{(1)} Q^{(1)}. \tag{3.4}$$

Inserting (3.4) into (3.3), we now find that the sampling matrix is

$$\begin{aligned} Y^{(2)} &= G^{(1)} Q^{(1)}\left(A^{(1)} - R^{(1)}\right) = \{\text{Use (3.2)}\} \\ &= G^{(1)} A^{(0)} P^{(1)} - G^{(1)} Q^{(1)} R^{(1)} = \{\text{Use (3.1)}\} = Y^{(1)} P^{(1)} - G^{(1)} Q^{(1)} R^{(1)}. \end{aligned} \tag{3.5}$$

Evaluating formula (3.5) is inexpensive since the first term is a permutation of the columns of the sampling matrix $Y^{(1)}$ and the second term is a product of thin matrices (recall that $Q^{(1)}$ is a product of b Householder reflectors).

Remark 3. Since the probability distribution for Gaussian random matrices is invariant under unitary maps, the formula (3.4) appears quite safe. After all, $G^{(1)}$ is Gaussian, and $Q^{(1)}$ is just a sequence of reflections, so it might be tempting to conclude that the new randomizing matrix must be Gaussian too. However, the matrix $Q^{(1)}$ unfortunately depends on the draw of $G^{(1)}$, so this argument does not work. Nevertheless, the dependence of $Q^{(1)}$ on $G^{(1)}$ is very subtle since this $Q^{(1)}$ is dictated primarily by the directions of the good pivot columns. Extensive practical experiments (see, e.g., section 4.2) indicate that the pivoting strategy described in this section based on downdating is just as good as the one that uses "pure" Gaussian matrices that was described in section 3.1.

3.3. Detailed description of the downdating procedure. Having described the downdating procedure informally in section 3.2, we in this section provide a detailed description using the notation for HQR that we used in section 3. First, let us assume that one iteration of the blocked algorithm has completed, so that, at the bottom of the loop body, the matrix A contains

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} U\backslash R_{11} & R_{12} \\ \hline U_{21} & \widehat{A}_{22} - U_{21}W_{12} \end{array}\right).$$

Here $\widehat{A}$ denotes the original contents of AP_1, where P_1 captures how columns have been swapped so far. Hence (cf. (3.2)),

$$\underbrace{\left(I - \left(\frac{U_{11}}{U_{21}}\right) T_{11}^{-H} \left(\frac{U_{11}}{U_{21}}\right)^{H}\right)}_{\left(\, Q_1 \middle| Q_2 \,\right)} \left(\begin{array}{c|c} \widehat{A}_{11} & \widehat{A}_{12} \\ \hline \widehat{A}_{21} & \widehat{A}_{22} \end{array}\right) P_1 = \left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & A_{22} \end{array}\right).$$

Now, let $\widetilde{G}_2$ be the next sampling matrix and $\widetilde{Y}_2 = \widetilde{G}_2 A_{22}$. In order to show how this new sampling matrix can be computed by downdating the last sampling matrix, consider that

$$\left(\, \widetilde{Y}_1 \middle| \widetilde{Y}_2 \,\right) = \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & A_{22} \end{array}\right)$$

for some matrix $\widetilde{G}_1$ and that

$$\begin{aligned}
(3.6)\quad \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & A_{22} \end{array}\right) &= \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & A_{22} \end{array}\right) - \left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & 0 \end{array}\right) \right) \\
&= \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & A_{22} \end{array}\right) - \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & 0 \end{array}\right) \\
&= \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\, Q_1 \middle| Q_2 \,\right) \widehat{A} P_1 - \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) \left(\, Q_1 \middle| Q_2 \,\right) \left(\, Q_1 \middle| Q_2 \,\right)^{H} \left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & 0 \end{array}\right).
\end{aligned}$$

The choice of randomizing matrix analogous to (3.4) is now

$$(3.7)\qquad \left(\, \widetilde{G}_1 \middle| \widetilde{G}_2 \,\right) = \left(\, G_1 \middle| G_2 \,\right) \left(\, Q_1 \middle| Q_2 \,\right).$$

Inserting the choice (3.7) into (3.6), we obtain

$$\left(\, \widetilde{Y}_1 \middle| \widetilde{Y}_2 \,\right) = \underbrace{\left(\, G_1 \middle| G_2 \,\right) \widehat{A}}_{\left(\, Y_1 \middle| Y_2 \,\right)} P_1 - \left(\, G_1 \middle| G_2 \,\right) \left(I - \left(\frac{U_{11}}{U_{21}}\right) T_{11}^{-H} \left(\frac{U_{11}}{U_{21}}\right)^{H} \right)^{H} \left(\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & 0 \end{array}\right).$$

Letting $\left(\overline{Y}_1 \middle| \overline{Y}_2 \right) = \left(Y_1 \middle| Y_2 \right) P_1$ we conclude that

$$\begin{aligned}\widetilde{Y}_2 &= \overline{Y}_2 - \left(G_1 \middle| G_2 \right) \left(I - \left(\frac{U_{11}}{U_{21}} \right) T_{11}^{-1} \left(\frac{U_{11}}{U_{21}} \right)^H \right) \left(\frac{R_{12}}{0} \right) \\ &= \overline{Y}_2 - \left(G_1 \middle| G_2 \right) \left(\left(\frac{R_{12}}{0} \right) - \left(\frac{U_{11}}{U_{21}} \right) T_{11}^{-1} {U_{11}}^H R_{12} \right) \\ &= \overline{Y}_2 - \left(G_1 - (G_1 U_{11} + G_2 U_{21}) T_{11}^{-1} {U_{11}}^H \right) R_{12},\end{aligned}$$

which can then be used to downdate the sampling matrix Y.

3.4. The blocked algorithm. In Figure 3, we give the blocked algorithm that results when the randomized pivot selection strategy described in section 3.1 is combined with the downdating techniques described in section 3.3. In that figure, there is a call to a function "HRQP_UNB" which is an unblocked HQR algorithm with column pivoting. The purpose of this call is to factor the current column panel so that the diagonal elements within blocks on the diagonal of R are ordered from largest to smallest in magnitude. Moreover, the call to "UpdatePivotInfo" takes the pivoting that occurred within the current panel (to ensure strictly decreasing diagonal elements within the current diagonal block of R_{11}) and merges this with the pivot information that occurred when determining the columns to be moved into that current panel.

3.5. Asymptotic cost analysis. In analyzing the asymptotic complexity of the method, we consider a matrix of size $m \times n$, with $m \geq n$. We assume that the block size b and the oversampling parameter p are kept fixed as m and n grow. We first note that all steps in Figure 3 highlighted in gray are part of the blocked HQR algorithm, which is known to have an asymptotic cost of $2mn^2 - 2/3n^3$ flops. This leaves us to discuss the overhead related to the other operations.

- $G := \textsc{rand\_iid}(b + p, m)$: Cost: ignored.
- $Y := GA$: Cost: $O((b + p)mn)$ flops.
- $s_1 := \textsc{DeterminePivots}(\left(Y_1 \middle| Y_2 \right), b)$: Cost: $O(b(b+p)(n-kb))$ flops during the kth iteration of the blocked algorithm, for a total of $O((b + p)n^2)$ flops. (Recall that the factorization of this matrix can stop after the first b columns have been identified.)
- $\cdots := \textsc{SwapCols}(s_1, \cdots)$: Cost: ignored.
- $Y_2 := Y_2 - \left(G_1 - (G_1 U_{11} + G_2 U_{21}) T_{11}^{-1} {U_{11}}^H \right) R_{12}$: Aggregate cost: $O((b + p)n^2)$.

Thus, the overhead is $O((b + p)(n^2 + mn))$ flops and the total cost is

$$2mn^2 - 2/3n^3 + O((b + p)(n^2 + mn)) \text{ flops},$$

which, asymptotically, approaches the same $2mn^2 - 2/3n^3$ cost as unpivoted HQR.

4. Experiments. This section describes the results from two sets of experiments. Section 4.1 compares the computational speed of the proposed scheme to existing state-of-the-art methods for computing column pivoted QR factorizations. Section 4.2 investigates how well the proposed randomized technique works at selecting pivot columns. Specifically, we investigate how well the rank-k truncated QR factorization approximates the original matrix and compare the results to those obtained by classical column pivoting.

4.1. Performance experiments. We have implemented the proposed HQRRP algorithm using the `libflame` [35, 34] library that allows our implementations to closely resemble the algorithms as presented in the paper.

```
Algorithm: $[A, T, s] := \text{HQRP\_RANDOMIZED\_BLK}(A, T, s, b, p)$
$G := \text{RAND\_IID}(b + p, n(A))$
$Y := GA$
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $T \rightarrow \left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right)$, $s \rightarrow \left(\begin{array}{c} s_T \\ \hline s_B \end{array}\right)$, $Y \rightarrow \left(\begin{array}{c|c} Y_L & Y_R \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$, $T_T$ has 0 rows, $s_T$ has 0 rows, $Y_L$ has 0 columns

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

    Determine block size $b \rightarrow \min(b, n(A_{BR}))$
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$, $\left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline T_2 \end{array}\right)$,
        $\left(\begin{array}{c} s_T \\ \hline s_B \end{array}\right) \rightarrow \left(\begin{array}{c} s_0 \\ \hline s_1 \\ \hline s_2 \end{array}\right)$, $\left(\begin{array}{c|c} Y_L & Y_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} Y_0 & Y_1 & Y_2 \end{array}\right)$
            where $A_{11}$ is $b \times b$, $T_1$ has $b$ rows, $s_1$ has $b$ rows, $Y_1$ has $b$ columns

    $s_1 := \text{DeterminePivots}(\left(\begin{array}{c|c} Y_1 & Y_2 \end{array}\right), b)$
    $\left(\begin{array}{c|c} A_{01} & A_{02} \\ \hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) := \text{SwapCols}(s_1, \left(\begin{array}{c|c} A_{01} & A_{02} \\ \hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right))$
    $\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), T_1, s_1'\right] := \text{HQR P \_UNB}(\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), T_1)$
    $A_{01} := \text{SwapCols}(s_1', A_{01})$
    $s_1 := \text{UpdatePivotInfo}(s_1', s_1)$
    $W_{12} := T_1^{-H}(U_{11}^H A_{12} + U_{21}^H A_{22})$
    $\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} A_{12} - U_{11} W_{12} \\ \hline A_{22} - U_{21} W_{12} \end{array}\right)$
    $\left(\begin{array}{c|c} Y_1 & Y_2 \end{array}\right) := \text{SwapCols}(s_1, \left(\begin{array}{c|c} Y_1 & Y_2 \end{array}\right))$
    $Y_2 := Y_2 - \left(G_1 - (G_1 U_{11} + G_2 U_{21}) T_{11}^{-1} {U_{11}}^H\right) R_{12}.$

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$, $\left(\begin{array}{c} T_T \\ \hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline T_2 \end{array}\right)$,
        $\left(\begin{array}{c} s_T \\ \hline s_B \end{array}\right) \leftarrow \left(\begin{array}{c} s_0 \\ \hline s_1 \\ \hline s_2 \end{array}\right)$, $\left(\begin{array}{c|c} Y_L & Y_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} Y_0 & Y_1 & Y_2 \end{array}\right)$

endwhile
```

FIG. 3. *Blocked Householder transformation based QR factorization with column pivoting based on randomization. In this algorithm, U_{11} equals the unit lower triangiular matrix stored below the diagonal of A_{11}, $U_{21} = A_{21}$, and $R_{12} = A_{12}$. The steps highlighted in gray constitute the blocked QR factorization without column pivoting from Figure* 2.

Platform details. All experiments reported in this article were performed on an Intel Xeon E5-2695 v3 (Haswell) processor (2.3 GHz), with 14 cores. In order to be able to show scalability results, the clock speed was throttled at 2.3 GHz, turning off so-called turbo boost. Each core can execute 16 double precision floating point operations per cycle, meaning that the peak performance of one core is 36.8 GFLOPS (billions of floating point operations per second). For comparison, on a single core, `dgemm` achieves around 33.6 GFLOPS. Other details of interest include that the OS used was Linux (Version 2.6.32-504.el6.x86_64), the code was compiled with gcc (Version 4.4.7), `dgeqrf` and `dgeqp3` were taken from LAPACK (Release 3.4.0), and the implementations were linked to BLAS from Intel's MKL library (Version 11.2.3). Our implementations were coded with `libflame` (Release 11104).

Implementations. We report performance for four implementations:

`dgeqrf`**.** The implementation of blocked HQR that is part of the `netlib` implemenation of LAPACK, modified so that the block size can be controlled.

`dgeqp3`**.** The implementation of blocked HQRP that is part of the `netlib` implemenation of LAPACK, based on [31], modified so that the block size can be controlled.

`HQRRPbasic`**.** Our implementation of HQRRP that computes new matrices G and Y in every iteration. This implementation deviates from the algorithm in Figure 3 in that it also incorporates additional column pivoting within the call to HQRRP_UNB.

`HQRRP`**.** The implementation of HQRRP that downdates Y. (It also includes pivoting within HQRRP_UNB).

`dgeqpx`**.** An implementation of HQRP with window pivoting [6, 5], briefly mentioned in the introduction. This algorithm consists of two stages. The first stage is a QR with window pivoting. An incremental condition estimator is employed to accept/reject the columns within a window. The size of the window is about twice the block size used by the algorithm to maintain locality. If all the columns in the window are unacceptable, all of them are rejected and fresh ones are brought into the window. The second stage is a postprocessing stage to validate the rank, that is, to check that all good columns are in the front, and all the bad columns are in the rear. This step is required because the window pivoting could fail to reveal the rank due to its shortsightedness (having only checked the window and having employed a cheap to compute condition estimator.) Sometimes, some columns must be moved between R_{11} that has been computed and matrices R_{12} and R_{22}, and then retriangularization must be performed with Givens rotations.

In all cases, we used algorithmic block sizes of $b = 64$ and 128. While likely not optimal for all problem sizes, these block sizes yield near best performance and, regardless, it allows us to easily compare and contrast the performance of the different implementations.

Results. As is customary in these kinds of performance comparisons, we compute the achieved performance as

$$\frac{4/3n^3}{\text{time (in sec.)}} \times 10^{-9} \text{ GFLOPS.}$$

Thus, even for the implementations that perform more computations, we only count the floating point operations performed by an unblocked HQR without pivoting.

Figure 4 reports performance on 1, 4, 8, and 14 cores. We see that `HQRRP` handily outperforms `dgeqp3` and to a lesser degree also outperforms `dgeqpx`. Moreover, the

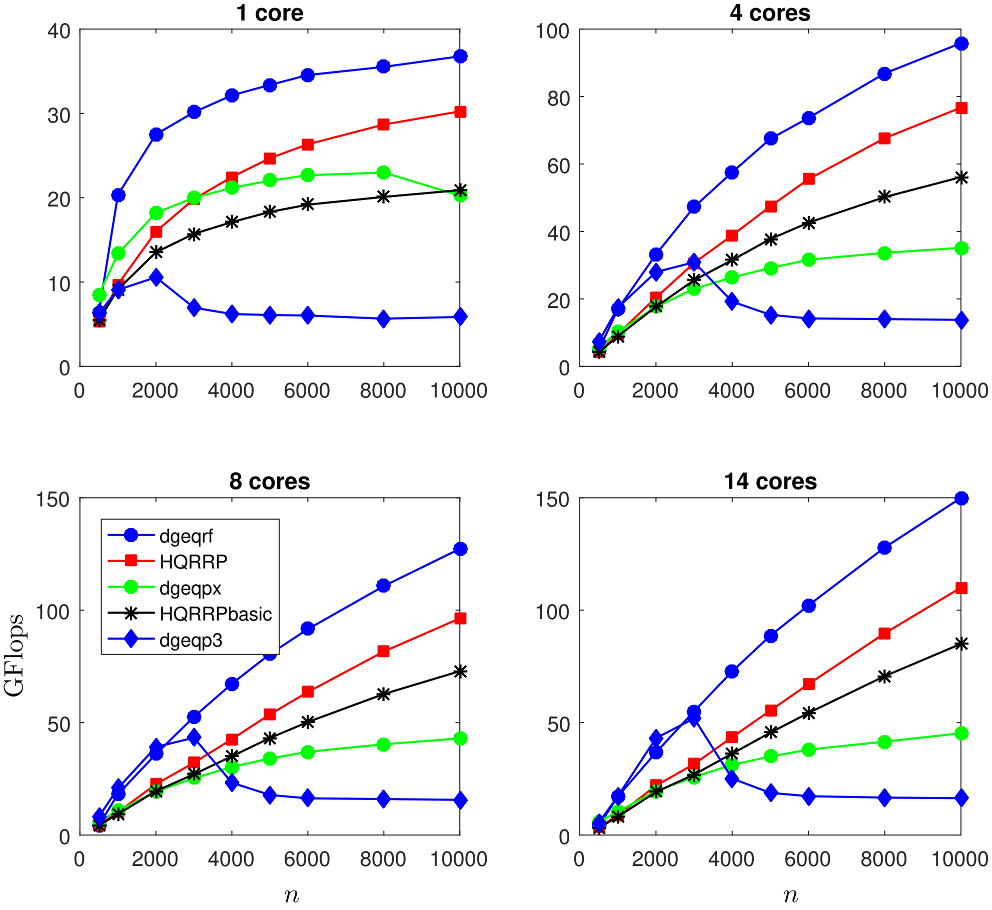

FIG. 4. *Computational speed (in standardized gigaflops) of the proposed randomized algorithm HQRRP for computing a column pivoted QR factorization of a matrix of size $n \times n$. The four graphs show results from test runs on* 1, 4, 8, *and* 14 *cores on an Intel Xeon* E5-2695 *v*3. *Observe that the scales on the vertical axes are different in the four graphs. For comparison, the graphs also show the times for competing algorithms (*`dgeqp3` *and* `dgeqpx`*) and for unpivoted QR (*`dgeqrf`*); see section* 4.1 *for details. The proposed algorithm HQRRP is faster than competing algorithms, with the gap growing as more cores are added. The block size for all algorithms was set to $b = 64$. Figure* 1 *shows the relative speeds in detail.*

asymptotic performance of `HQRRP` appears to approach that of `dgeqrf`, in particular for the single core case. We also see that while the relative performances of all five methods remain qualitatively the same across the four graphs, it is clear that as the number of cores grows, the speed advantage of `HQRRP` over `dgeqp3` becomes even further pronounced; cf. Figure 1.

Figure 4 also shows that while the absolute speed of all five algorithms studied improves as the number of cores grows, all five fall substantially short of ideal scaling (in that the number of gigaflops *per core* falls as the number of cores grows). This observation underscores the need for further research in this area.

In order to investigate the effect of the block size on the computational speed, we reran the experiments shown in Figure 4 with a block size of $b = 128$ instead of $b = 64$, with the results shown in Figure 5. We see that the choice $b = 64$ tends to lead to slightly faster execution, but the key take-away from this comparison is that

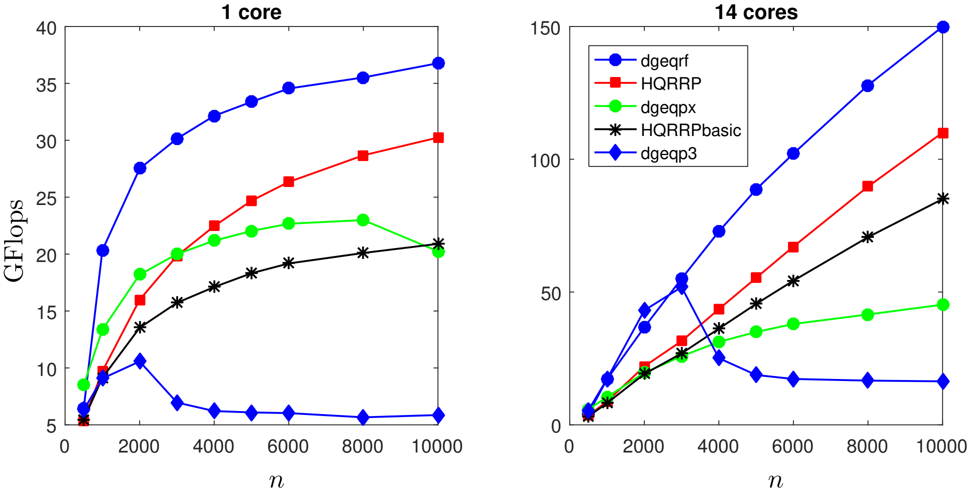

Fig. 5. *Computational speeds (in standardized gigaflops) for the same problem as that shown in Figure* 4. *In this figure, results are shown for a blocksize of* $b = 128$, *in contrast with the blocksize* $b = 64$ *that was used in Figure* 4.

the speed is relatively insensitive to the precise choice of block size (within reason, of course).

4.2. Quality experiments. In this section, we describe the results of numerical experiments that were conducted to compare the quality of the pivot choices made by our randomized algorithm HQRRP to those resulting from classical column pivoting. Specifically, we compared how well partial factorizations reveal the numerical ranks of four different test matrices:

- *Matrix* 1 *(fast decay):* This is an $n \times n$ matrix of the form $A = UDV^*$ where U and V are randomly drawn matrices with orthonormal columns (obtained by performing QR on a random Gaussian matrix) and where D is diagonal with entries given by $d_j = \beta^{(j-1)/(n-1)}$ with $\beta = 10^{-5}$.
- *Matrix* 2 *(S shaped decay):* This matrix is built in the same manner as Matrix 1, but now the diagonal entries of D are chosen to first hover around 1, then decay rapidly, and then level out at 10^{-6}, as shown later in Figure 7 (black line).
- *Matrix* 3 *(single layer BIE):* This matrix is the result of discretizing a boundary integral equation (BIE) defined on a smooth closed curve in the plane. To be precise, we discretized the so-called single layer operator associated with the Laplace equation using a sixth order quadrature rule designed by Alpert [1]. This is a well-known ill-conditioned operator for which column pivoting is essential in order to stably solve the corresponding linear system.
- *Matrix* 4 *(Kahan):* This is a variation of the "Kahan counterexample" [22] which is designed to trip up classical column pivoting. The matrix is formed as $A = SK$, where

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots \\ 0 & \zeta & 0 & 0 & \cdots \\ 0 & 0 & \zeta^2 & 0 & \cdots \\ 0 & 0 & 0 & \zeta^3 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} 1 & -\phi & -\phi & -\phi & \cdots \\ 0 & 1 & -\phi & -\phi & \cdots \\ 0 & 0 & 1 & -\phi & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

for some positive parameters ζ and ϕ such that $\zeta^2 + \phi^2 = 1$. In our experiments, we chose $\zeta = 0.99999$.

For each test matrix, we computed QR factorizations

$$AP = QR \tag{4.1}$$

using three different techniques:

- `HQRP`: The standard QR factorization `qr` built in to MATLAB R2015a.
- `HQRRPbasic`: The randomized algorithm described in Figure 3, but without the updating strategy for computing the sample matrix Y.
- `HQRRP:` The randomized algorithm described in Figure 3.

Our implementations of both `HQRRPbasic` and `HQRRP` deviate from what is shown in Figure 3 in that they also incorporate column pivoting within the call to HQRRP_UNB. In all experiments, we used test matrices of size $4\,000 \times 4\,000$, a block size of $b = 100$, and an oversampling parameter of $p = 5$.

As a quality measure of the pivoting strategy, we computed the errors e_k incurred when the factorization is truncated to its first k components. To be precise, these residual errors are defined via

$$e_k = \|AP - Q(:, 1:k)R(1:k, :)\| = \|R((k+1):n, (k+1):n)\|. \tag{4.2}$$

The results are shown in Figures 6–9, for the four different test matrices. The black lines in the graphs show the theoretically minimal errors incurred by a rank-k approximation. These are provided by the Eckart–Young theorem [11] which states that, with $\{\sigma_j\}_{j=1}^{n}$ denoting the singular values of A,

$$e_k \geq \sigma_{k+1} \qquad \text{when errors are measured in the spectral norm, and}$$

$$e_k \geq \left(\sum_{j=k+1}^{n} \sigma_j\right)^{1/2} \qquad \text{when errors are measured in the Frobenius norm.}$$

We observe in all cases that the quality of the pivots chosen by the randomized method very closely matches those resulting from classical column pivoting. The one exception is the Kahan counterexample ("Matrix 4"), where the randomized algorithm performs much better. (The importance of the last point should not be overemphasized since this example is designed specifically to be adversarial for classical column pivoting.)

When classical column pivoting is used, the factorization (4.1) produced has the property that the diagonal entries of R are strictly decaying in magnitude,

$$|R(1,1)| \geq |R(2,2)| \geq |R(3,3)| \geq \cdots .$$

When the randomized pivoting strategies are used, this property is not enforced. To illustrate this point, we show in Figure 10 the values of the diagonal entries obtained by the randomized strategies versus what is obtained with classical column pivoting.

5. Conclusions and future work.

We have described the algorithm HQRRP which is a blocked version of Householder QR with column pivoting. The main innovation compared to earlier work is that pivots are determined in groups using a technique based on randomized projections. We demonstrated that the quality of the chosen pivots is for practical purposes indistinguishable from traditional column pivoting (cf. Figures 6–8) and that the dominant term in the asymptotic flop count

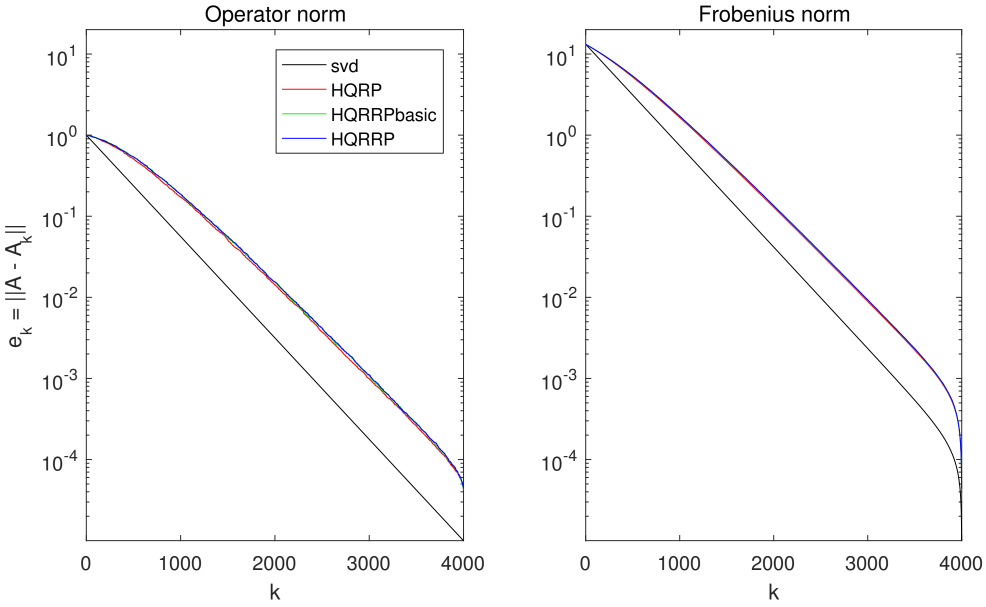

Fig. 6. *Residual errors e_k for "Matrix 1" as a function of the truncation rank k; cf.* (4.2). *The red line shows the results from traditional column pivoting, while the green and blue lines refer to the randomized methods we propose. The black line indicates the theoretically minimal errors resulting from a rank-k partial singular value decomposition.*

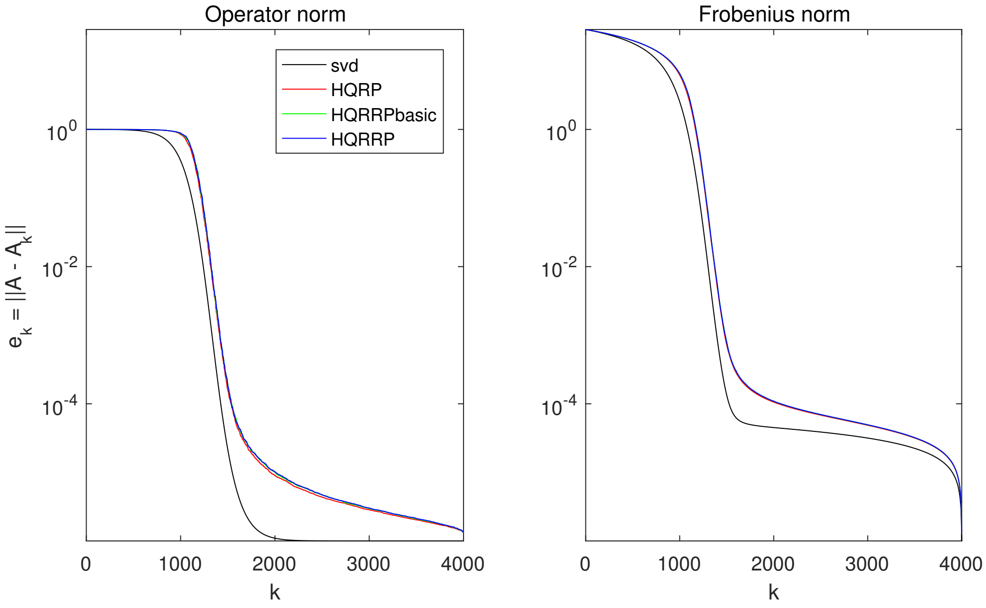

Fig. 7. *Residual errors e_k for "Matrix 2" as a function of the truncation rank k. Notation is the same as in Figure* 6.

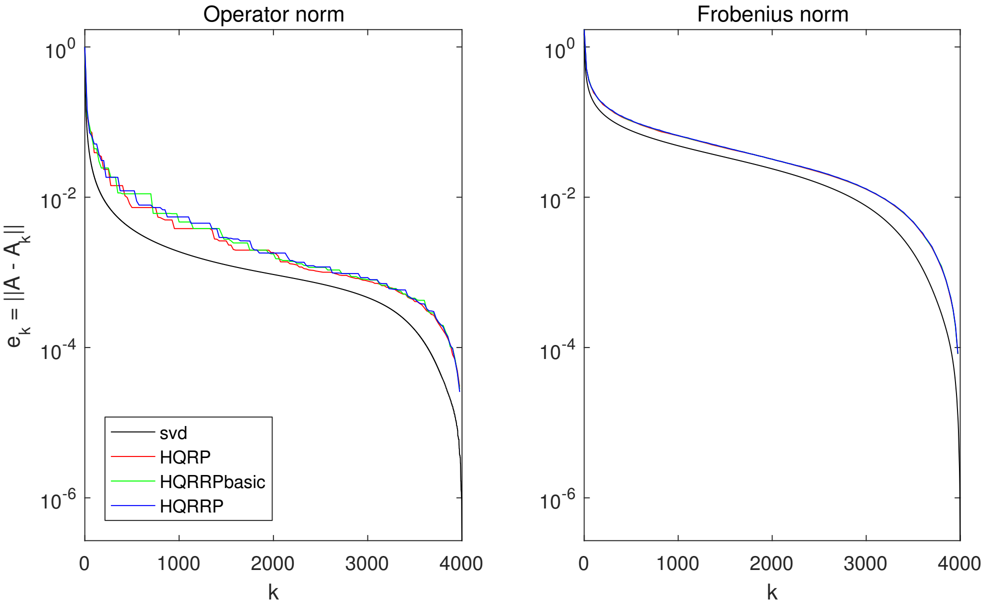

FIG. 8. *Residual errors e_k for "Matrix 3" (discretized boundary integral operator) as a function of the truncation rank k. Notation is the same as in Figure 6.*

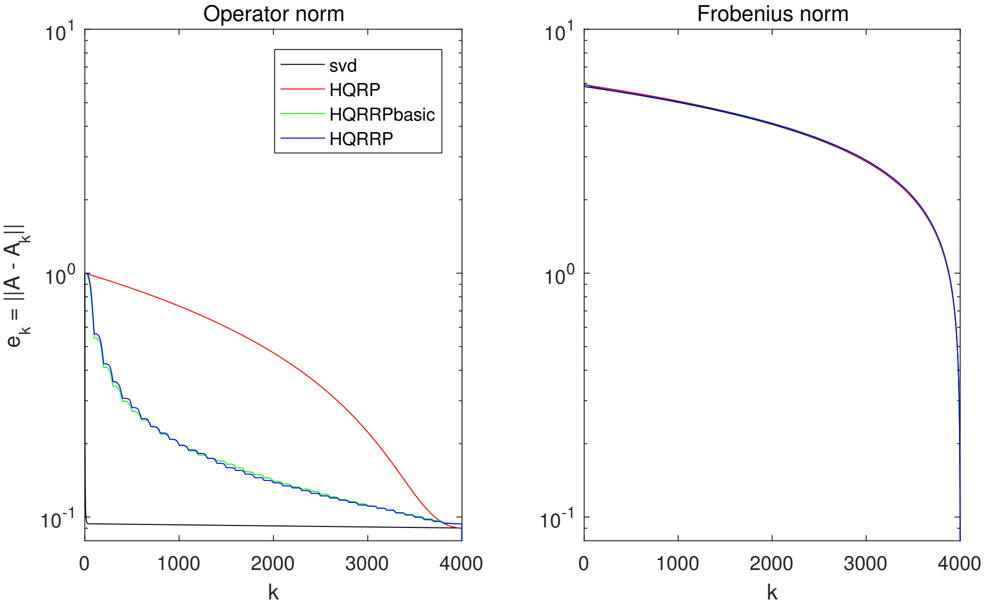

FIG. 9. *Residual errors e_k for "Matrix 4" (Kahan) as a function of the truncation rank k. Notation is the same as in Figure 6.*

equals that of nonpivoted QR. Importantly, we also demonstrated through numerical experiments that HQRRP is very fast, in fact almost as fast as unpivoted HQR.

The technique described opens up several potential avenues for future research. The speed gains we demonstrate on single core and shared memory multicore machines

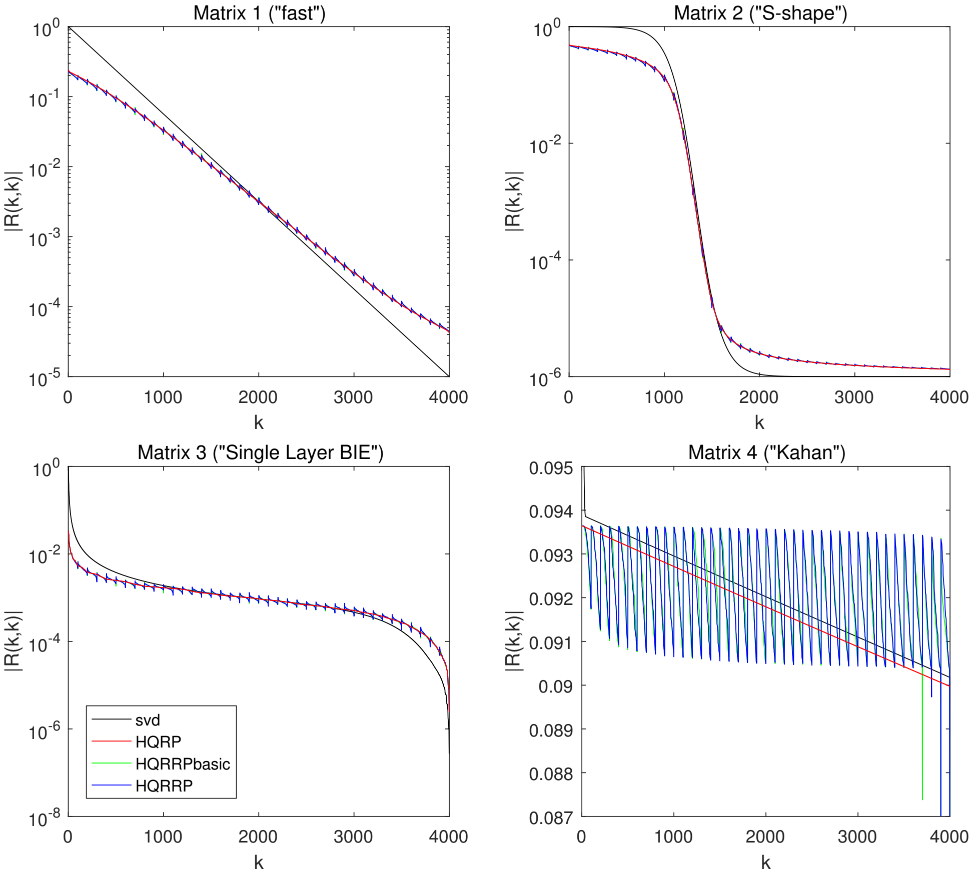

FIG. 10. *For each of the four test matrices described in section* 4.2*, we show the magnitudes of the diagonal entries in the "R"-factor in a column pivoted QR factorization. We compare classical column pivoting (red) with the two randomized techniques proposed here (blue and green). We also show the singular values of each matrix (black) for reference.*

is due to the reduction in data movement. Equivalently, data moved between memory layers is carefully amortized. We expect the technique described to have an even more pronounced advantage over traditional column pivoted QR when implemented in more severely communication constrained environments such as a matrix processed on a GPU or a distributed memory parallel machine, or stored out-of-core.

The randomized sampling techniques we describe can also be used to construct very close to optimal rank-revealing factorizations. To describe the idea, we note that a column pivoted QR factorization of a given matrix A can be written as

$$A = Q\,R\,P^{*}, \tag{5.1}$$

where Q is orthonormal, R is upper triangular, and P is a permutation matrix. In this manuscript, we used randomized sampling to determine the permutation matrix P. It turns out that for a modest additional cost, one can build a factorization that takes the form (5.1), but with both Q and P built as products of Householder reflectors. This generalization allows us to bring R not only to upper triangular form but very close to being diagonal, with accurate approximations to the singular values of A on its diagonal. This discovery was reported in [25] and is a subject of ongoing research.

How to scale the presented algorithm to very large numbers of cores is an open research question. Techniques such as "compute ahead" will have to be employed to ensure that the factorization of the current panel (A_{11} and A_{21}) and downdate of Y do not start dominating the parts of the computation that can be cast in terms of GEMM.

6. Software.

A number of implementations of the discussed algorithm are available under 3-clause (modified) BSD license from https://github.com/flame/hqrrp. Included are implementations that directly link to LAPACK [3] as well as implementations that use the `libflame` [30, 17] library. For those who use the LAPACK routine dgeqp3 routine, a plug compatible routine dgeqp4 is provided.

A distributed memory implementation of the algorithm has been incorporated into the Elemental software package by Poulson et al. [29], available at https://github.com/elemental/Elemental.

Acknowledgments. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] B. K. Alpert, *Hybrid Gauss-trapezoidal quadrature rules*, SIAM J. Sci. Comput., 20 (1999), pp. 1551–1584.

[2] *AMD Core Math Library*, http://developer.amd.com/tools-and-sdks/cpu-development/amd-core-math-library-acml/.

[3] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. D. Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.

[4] C. Bischof and C. Van Loan, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s2–s13.

[5] C. H. Bischof and G. Quintana-Ortí, *Algorithm* 782*: Codes for rank-revealing QR factorizations of dense matrices*, ACM Trans. Math. Software, 24 (1998), pp. 254–257.

[6] C. H. Bischof and G. Quintana-Ortí, *Computing rank-revealing QR factorizations of dense matrices*, ACM Trans. Math. Software, 24 (1998), pp. 226–253.

[7] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[8] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, 1991.

[9] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices.* II*. Computing a low-rank approximation to a matrix*, SIAM J. Comput., 36 (2006), pp. 158–183.

[10] J. Duersch and M. Gu, *True BLAS-*3 *Performance QRCP Using Random Sampling*, preprint, arXiv:1509.06820, 2015.

[11] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[12] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.

[14] K. Goto and R. A. Geijn, *Anatomy of high-performance matrix multiplication*, ACM Trans. Math. Software, 34 (2008), p. 12.

[15] K. Goto and R. A. van de Geijn, *On Reducing TLB Misses in Matrix Multiplication*, Tech. Report CS-TR-02-55, Department of Computer Sciences, The University of Texas at Austin, 2002.

[16] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.

[17] J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn, *FLAME: Formal Linear Algebra Methods Environment*, ACM Trans. Math. Software, 27 (2001), pp. 422–455, http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html.

[18] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[19] *Engineering and Scientific Subroutine Library*, http://www-03.ibm.com/systems/power/software/essl/.

[20] *Math Kernel Library*, http://developer.intel.com/software/products/mkl/.

[21] T. Joffrain, T. M. Low, E. S. Quintana-Ortí, R. van de Geijn, and F. G. Van Zee, *Accumulating Householder transformations, revisited*, ACM Trans. Math. Software, 32 (2006), pp. 169–179, https://doi.org/10.1145/1141885.1141886.

[22] W. Kahan, *Numerical linear algebra*, Canad. Math. Bull, 9 (1966), pp. 757–801.

[23] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172, https://doi.org/10.1073/pnas.0709640104.

[24] M. W. Mahoney, *Randomized algorithms for matrices and data*, Found. Trends Machine Learning, 3 (2011), pp. 123–224.

[25] P. Martinsson, *Blocked Rank-Revealing Gr Factorizations: How Randomized Sampling Can Be Used to Avoid Single-Vector Pivoting*, preprint, arXiv:1505.08115, 2015.

[26] P.-G. Martinsson, V. Rokhlin, and M. Tygert, *A Randomized Algorithm for the Approximation of Matrices*, Yale CS research report YALEU/DCS/RR-1361, Computer Science Department, Yale University, 2006.

[27] P.-G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the decomposition of matrices*, Appl. Comput. Harmon. Anal., 30 (2011), pp. 47–68, https://doi.org/10.1016/j.acha.2010.02.003.

[28] P.-G. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., 38 (2016), pp. S485–S507.

[29] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, *Elemental: A new framework for distributed memory dense matrix computations*, ACM Trans. Math. Softw., 39 (2013), p. 13.

[30] E. S. Quintana, G. Quintana, X. Sun, and R. van de Geijn, *A note on parallel matrix inversion*, SIAM J. Sci. Comput., 22 (2001), pp. 1762–1771.

[31] G. Quintana-Ortí, X. Sun, and C. H. Bischof, *A BLAS-3 version of the QR factorization with column pivoting*, SIAM J. Sci. Comput., 19 (1998), pp. 1486–1494, https://doi.org/10.1137/S1064827595296732.

[32] R. Schreiber and C. Van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53–57.

[33] G. Stewart, *Matrix Algorithms Volume 1: Basic Decompositions*, SIAM, Philadelphia, 1998.

[34] F. G. Van Zee, `libflame`*: The Complete Reference*, http://www.cs.utexas.edu/users/flame/web/FLAMEPublications.html, (2012).

[35] F. G. Van Zee, E. Chan, R. van de Geijn, E. S. Quintana-Ortí, and G. Quintana-Ortí, *The libflame library for dense matrix computations*, IEEE Comput. Sci. Eng., 11 (2009), pp. 56–62.

[36] F. G. Van Zee and R. A. van de Geijn, *BLIS: A framework for rapidly instantiating BLAS functionality*, ACM Trans. Math. Software, 41 (2015).

[37] S. Voronin and P. Martinsson, *A CUR Factorization Algorithm Based on the Interpolative Decomposition*, arXiv:1412.8447, 2014.

[38] D. P. Woodruff, *Sketching as a Tool for Numerical Linear Algebra*, preprint, arXiv:1411.4357, 2014.

[39] X. Zhang, Q. Wang, and Y. Zhang, *Model-driven level 3 BLAS performance optimization on Loongson 3a processor*, in Proceedings of the 18th International Conference on Parallel and Distributed Systems, IEEE Computer Society, 2012, pp. 684–691.