Parallel Out-of-Core Computation and Updating of the QR Factorization

UTCS Technical Report TR-03-40

PLAPACK Working Note #13

Brian C. Gunter
Center for Space Research
The University of Texas at Austin
Austin, TX 78712
gunter@csr.utexas.edu

Robert A. van de Geijn
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
rvdg@cs.utexas.edu

September 10, 2003

Abstract

This paper discusses the high-performance parallel implementation of the computation and updating of QR factorizations of dense matrices, including problems large enough to require out-of-core computation, where the matrix is stored on disk. The algorithms presented here are scalable both in problem size and as the number of processors increases. Implementation using the Parallel Linear Algebra Package (PLAPACK) and the Parallel Out-of-Core Linear Algebra Package (POOCLAPACK) is discussed. The methods are shown to attain excellent performance, in some cases attaining roughly 80% of the peak of the architectures on which the experiments were performed.

1 Introduction

With the recent improvements in memory access, storage capacity, and processing power of high-performance computers, operating on large dense linear systems is not as daunting a task as it has been in the past. One such realm where this new capability has been of extreme value is in the Earth sciences, where large overdetermined Linear Least-Squares problems are encountered which can involve tens of thousands of parameters and millions of observations [16]. Since these systems have the potential to be mildly ill-conditioned, the method chosen to compute the least-squares solution is one utilizing the QR factorization, since it provides greater accuracy than the Method of Normal Equations. To tackle a problem of this size requires the use of an efficient and scalable implementation of the QR factorization that can take advantage of the power of modern day supercomputers. Initially, an in-core parallel implementation was developed, but later an out-of-core (OOC) implementation was created to handle even larger problems.

In reporting the results of our research, this paper makes the following contributions:

- It reviews the standard techniques for the high-performance implementation of the QR factorization based on the Householder transformation [4, 3, 13, 8].
- It extends these techniques to the problem of updating the QR factorization as additional batches of observations are added to the system.
- It demonstrates how the techniques used to update a QR factorization can be used to implement an OOC QR factorization algorithm that is more scalable than other previously proposed OOC approaches

for this problem [7]. In particular, while it was previously observed that so-called tiled approaches are more scalable than so-called "slab" approaches for the OOC computation of the Cholesky factorization [31, 26, 25, 17] and a (possibly unstable) variant of the LU factorization [27], only (nonscalable) slab approaches had been previously proposed for the OOC QR factorization.

- It discusses a parallel implementation of the algorithms using the Parallel Linear Algebra Package (PLAPACK) [32] and its out-of-core extension, the Parallel Out-of-Core Linear Algebra Package (POOCLAPACK) [26, 25, 17, 1].
- It reports excellent performance attained on massively parallel distributed memory supercomputers.

While we reported initial performance results for the implementations in a previous (conference) paper [19], this paper goes into considerably more depth.

This paper is structured as follows: Section 2 briefly describes the notation used in the paper. Section 3 reviews the QR factorization using Householder transformations. This includes a discussion of the block algorithm as well as the QR factorization's application to the least squares problem. Section 4 discusses how the standard QR factorization can be updated when new information becomes available. This technique later becomes a key component of the OOC algorithm, which is given in Section 5. The actual parallel implementation is discussed in Section 6. Results from experiments are reported in Section 7. Section 8 provides some final thoughts and conclusions.

2 Notation

In this paper, we adopt the following conventions: Matrices, vectors, and scalars are denoted by upper-case, lower-case, and lower-case Greek letters, respectively. The identity matrix will be denoted by I and e_1 will denote the first column of the identity matrix (in other words, the vector with first element equal to unity and all other elements equal to zero). The dimensions and lengths of such matrices and vectors will generally be obvious from context.

Algorithms in this paper are given in a notation that we have recently adopted as part of the FLAME project [15, 24]. If one keeps in mind that the double lines in the partitioned matrices and vectors relate to how far into the matrices and vectors the computation has proceeded, we believe the notation to be intuitively obvious. If not, we suggest that the reader consult some of these related papers.

3 Computing the QR factorization via Householder Transformations

Given an $m \times n$ real-valued matrix A, with $m \geq n$, the QR factorization is given by

$$A = QR$$

where the $m \times m$ matrix Q has mutually orthonormal columns ($Q^T Q = I$) and the $m \times n$ matrix R is upper triangular.

There are many different methods for computing the QR factorization, including those based on Givens rotations, orthogonalization via Gram-Schmidt and Modified Gram-Schmidt, and Householder transformations [13, 33]. For dense matrices, the method of choice depends largely on how the factorization is subsequently used, the stability of the system, and the dimensions of the matrix. For problems where $m \gg n$, the method based on Householder transformations is typically the algorithm of choice, especially when Q needs not be explicitly computed.

3.1 Householder transformations (reflectors)

Given the real-valued vector x of length m, partition

$$x = \left(\frac{\chi_1}{x_2} \right)$$

```
Partition $A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$ and $b \rightarrow \left(\begin{array}{c} b_T \\ \hline\hline b_B \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$ and $b_T$ has 0 elements

while $n(A_{BR}) \neq 0$ do
    Repartition
        $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & a_{01} & A_{02} \\ \hline\hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} b_T \\ \hline\hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline\hline \beta_1 \\ \hline b_2 \end{array}\right)$
        where $\alpha_{11}$ and $\beta_1$ are scalars

    $\left[\left(\begin{array}{c} 1 \\ \hline u_2 \end{array}\right), \eta, \beta_1\right] := h\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$
    $\alpha_{11} := \eta$
    $a_{21} := u_2$
    $w^T := a_{12}^T + u_2^T A_{22}$
    $\left(\begin{array}{c} a_{12}^T \\ \hline A_{22} \end{array}\right) := \left(\begin{array}{c} a_{12}^T - \beta_1 w^T \\ \hline A_{22} - \beta_1 u_2 w^T \end{array}\right)$

    Continue with
        $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline\hline A_{20} & a_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} b_T \\ \hline\hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline\hline b_2 \end{array}\right)$
enddo
```

Figure 1: Unblocked Householder QR factorization.

where χ_1 equals the first element of x.

The Householder vector associated with x is defined as the vector

$$u = \left(\begin{array}{c} 1 \\ \hline x_2/\nu_1 \end{array}\right),$$

where $\nu_1 = \chi_1 + \text{sign}(\chi_1)\|x\|_2$. If $\beta = \frac{2}{u^T u}$ then $(I - \beta u u^T)x = \eta e_1$, annihilating all but the first element of x. Here $\eta = -\text{sign}(\chi_1)\|x\|_2$. The transformation $I - \beta u u^T$, with $\beta = 2/u^T u$ is referred to as a Householder transformation or reflector.

Let us introduce the notation $[u, \eta, \beta] := h(x)$ as the computation of the above mentioned η, u, and β from vector x and the notation $H(x)$ for the transformation $(I - \beta u u^T)$ where $[u, \eta, \beta] = h(x)$. An important feature of $H(x)$ is that it is orthonormal ($H(x)^T H(x) = H(x)H(x)^T = I$) and symmetric ($H(x)^T = H(x)$).

3.2 A simple algorithm for the QR factorization via Householder transformations

The computation of the QR factorization commences as described in Figure 1. The idea is that Householder transformations are computed to successively annihilate elements below the diagonal of matrix A. The Householder vectors are stored below the diagonal over the elements of A that have been so annihilated. Upon completion, matrix R has overwritten the upper triangular part of the matrix, while the Householder vectors are stored in the lower trapezoidal part of the matrix. The scalars β discussed above are stored in the vector b.

If the matrix Q is explicitly desired, it can be formed by computing the first n columns of $H_1 H_2 \cdots H_n$ where H_i equals the ith Householder transformation computed as part of the factorization described above. In our application, we do not need to form Q explicitly and thus will not discuss the issue further.

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

while $n(A_{BR}) \neq 0$ do
    Determine block size $k$
    Repartition
        $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline\hline T_1 \\ \hline T_2 \end{array}\right)$
        where $A_{11}$ is $k \times k$ and $T_1$ has $k$ rows
    ====================================================================
    $\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), \eta, b_1\right] := \left[\left(\begin{array}{c} \{Y\backslash R\}_{11} \\ \hline Y_{21} \end{array}\right), \eta, b_1\right] = QR\left(\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)\right)$
    Compute $T_1$ from $\left[\left(\begin{array}{c} \{Y\backslash R\}_{11} \\ \hline Y_{21} \end{array}\right), \eta, b_1\right]$
    $\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := \left(I + \left(\begin{array}{c} Y_{11} \\ \hline Y_{21} \end{array}\right) T_1^T \left( \begin{array}{c|c} Y_{11}^T & Y_{21}^T \end{array}\right)\right) \left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right)$
    ====================================================================
    Continue with
        $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline\hline T_2 \end{array}\right)$
enddo
```

Figure 2: Blocked Householder QR factorization.

3.3 A high-performance (blocked) algorithm for the QR factorization

It is well-known that high performance can be achieved in a portable fashion by casting algorithms in terms of matrix-matrix multiplication [2, 10, 12]. We now review how to do so for the QR factorization [3, 2].

Two observations play a key role:

- Let $u_1, \ldots, u_k$ equal the first k Householder vectors computed as part of the factorization and $\beta_1, \ldots, \beta_k$ the corresponding scalars. Then $H_k \cdots H_2 H_1 = I + YTY^T$ where Y is a $n \times k$ unit lower-trapezoidal matrix, T is a $k \times k$ upper-triangular matrix, and the jth column of the lower-trapezoidal part of Y equals u_j.
- The QR factorization of the first k columns of A yields the same k vectors u_k and same values in the upper triangular part of those k columns as would a full QR factorization.

Notice that this suggests the blocked algorithm given in Figure 2.

Note 1 *Notice that the algorithm stores the "T" matrices that are part of the block Householder transformation $I + YTY^T$. This is a distinct departure from traditional implementations like those found in LAPACK [2] and will allow us to avoid recomputation of those matrices as part of the OOC implementation of the QR factorization.*

3.4 Solving multiple Linear Least-Squares problems

Given a real-valued $m \times n$ matrix A and vector y of length m, the linear least-squares problem is generally stated as

$$\min_x \|y - Ax\|_2$$

where the desired result is a vector x that minimizes the above expression. It is well-known that the minimizing vector x can be found by computing the QR factorization $A = QR$, computing $z = Q^T y$, and solving $Rx = z_T$ where z_T denotes the first n elements of z.

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

**while** $n(A_{BR}) \neq 0$ **do**
    **Determine block size** $k$
    **Repartition**
        $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$,
        $\left(\begin{array}{c} B_T \\ \hline\hline B_B \end{array}\right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline\hline B_1 \\ \hline B_2 \end{array}\right)$, and $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline\hline T_1 \\ \hline T_2 \end{array}\right)$
            **where** $A_{11}$ is $k \times k$ and $B_1$ and $T_1$ have $k$ rows

    $\left(\begin{array}{c} B_1 \\ \hline B_2 \end{array}\right) := \left(I + \left(\begin{array}{c} Y_{11} \\ \hline Y_{21} \end{array}\right) T_1^T \left(\begin{array}{c|c} Y_{11}^T & Y_{21}^T \end{array}\right)\right) \left(\begin{array}{c} B_1 \\ \hline B_2 \end{array}\right)$

        NOTE: Here $Y_{11}$ refers to the lower triangular part of $A_{11}$ and $Y_{21}$ to $A_{21}$

    **Continue with**
        $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$,
        $\left(\begin{array}{c} B_T \\ \hline\hline B_B \end{array}\right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline B_1 \\ \hline\hline B_2 \end{array}\right)$, and $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline\hline T_2 \end{array}\right)$
**enddo**
```

Figure 3: Forward substitution-like of right-hand-side matrix B.

Alternatively, one can think of this as follows: Append y to A to form $\left(\begin{array}{c|c} A & y \end{array} \right)$. Compute the QR factorization $A = QR$, storing the Householder vectors and R over A. Update y by applying the Householder transformations used to compute R to vector y, which overwrites y with z. Finally, solve $Rx = z_T$ with the first n elements of the updated y. This second approach is reminiscent of how a linear system can be solved by appending the right-hand-side vector to the system and performing an LU factorization (or, equivalently, Gaussian elimination) on the appended system, followed by a back-substitution step.

Finally, if there exists a set of right-hand-sides, one can simultaneously solve a linear least-squares problem with A and columns of B by the following approach: Append B to A to form $\left(\begin{array}{c|c} A & B \end{array} \right)$. Compute the QR factorization $A = QR$, storing the Householder vectors and R over A. Update B by applying the Householder transformations used to compute R to matrix B, which overwrites B with Z. Finally, solve $RX = Z_T$ with the first n rows of the updated B. It is this second operation with a right-hand-side B that we will encounter in the OOC implementation of the QR factorization. An algorithm for the first, forward substitution-like, step is given in Figure 3.

Note 2 *Notice that because we store the "T" matrices that are part of the block Householder transformation $I + YTY^T$, they need not be recomputed as part of the "forward substitution" step on matrix B. This is again a distinct departure from traditional implementations like those found in LAPACK.*

4 Updating the QR Factorization

Frequently, the linear equations used in the least squares problem are collected incrementally. For example, if the observations from a particular instrument are only collected or contributed on a monthly basis, it would be useful to combine each new batch of data into the existing solution without having to recombine all of the

```
Partition $R \rightarrow \left(\begin{array}{c||c} R_{TL} & R_{TR} \\ \hline\hline R_{BL} & R_{BR} \end{array}\right)$, $C \rightarrow \left(\begin{array}{c||c} C_L & C_R \end{array}\right)$, and $b \rightarrow \left(\begin{array}{c} b_T \\ \hline\hline b_B \end{array}\right)$
    where  $R_{TL}$ and $C_L$ are $0 \times 0$ and $b_T$ has 0 elements

while $n(R_{BR}) \neq 0$ do
    Repartition
        $\left(\begin{array}{c||c} R_{TL} & R_{TR} \\ \hline\hline R_{BL} & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} R_{00} & r_{01} & R_{02} \\ \hline\hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right)$,
        $\left(\begin{array}{c||c} C_L & C_R \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} C_0 & c_1 & C_2 \end{array}\right)$, and $\left(\begin{array}{c} b_T \\ \hline\hline b_B \end{array}\right) \rightarrow \left(\begin{array}{c} b_0 \\ \hline\hline \beta_1 \\ \hline b_2 \end{array}\right)$
            where  $\rho_{11}$ and $\beta_1$ are scalars and $c_1$ is a column
    ------------------------------------------------------------
    $\left[\left(\frac{1}{u_2}\right), \eta, \beta_1\right] := h\left(\frac{\rho_{11}}{c_1}\right)$
    $\rho_{11} := \eta$
    $c_1 := u_2$
    $w^T := r_{12}^T + u_2^T C_2$
    $\left(\frac{r_{12}^T}{C_2}\right) := \left(\frac{r_{12}^T - \beta_1 w^T}{C_2 - \beta_1 u_2 w^T}\right)$
    ------------------------------------------------------------
    Continue with
        $\left(\begin{array}{c||c} R_{TL} & R_{TR} \\ \hline\hline R_{BL} & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline\hline R_{20} & r_{21} & R_{22} \end{array}\right)$,
        $\left(\begin{array}{c||c} C_L & C_R \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} C_0 & c_1 & C_2 \end{array}\right)$, and $\left(\begin{array}{c} b_T \\ \hline\hline b_B \end{array}\right) \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline\hline b_2 \end{array}\right)$
enddo
```

Figure 4: Unblocked update to a QR factorization.

previous data again. In other words, it would be nice to update the QR factorization rather than compute it from the original data to which the new data has been added. We now show how the QR factorization can be updated as additional batches of equations (i.e., observations) become available.

4.1 Factorization

Let us assume that we have computed Q and R such that $A = QR$, overwriting A with the Householder vectors and upper triangular matrix R, and storing the "T" matrices in matrix T. Thus, we have available $A = \{Y \backslash R\}$ and T. Now, consider the QR factorization of matrix

$$\left(\begin{array}{c} A \\ C \end{array}\right) = \bar{Q}\bar{R} \tag{1}$$

A key observation is that the QR factorization of

$$\left(\begin{array}{c} R \\ C \end{array}\right) \tag{2}$$

produces the same upper triangular matrix $\bar{R}$ as does the factorization in (1). If we are not interested in explicitly forming $\bar{Q}$ and are satisfied with storing the Householder vectors required to first compute the QR factorization of A and next the QR factorization in (2), then we have an approach for computing the QR factorization of an updated system. The unblocked and blocked algorithm for doing so is given in Figures 4 and 5, respectively.

```
Partition $R \rightarrow \left(\begin{array}{c||c} R_{TL} & R_{TR} \\ \hline\hline R_{BL} & R_{BR} \end{array}\right)$, $C \rightarrow \left(\begin{array}{c||c} C_L & C_R \end{array}\right)$, and $T \rightarrow \left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right)$
    where $R_{TL}$ and $C_L$ are $0 \times 0$ and $T_T$ has 0 rows

while $n(R_{BR}) \neq 0$ do
    Determine block size $k$
    Repartition
        $\left(\begin{array}{c||c} R_{TL} & R_{TR} \\ \hline\hline R_{BL} & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} R_{00} & R_{01} & R_{02} \\ \hline\hline R_{10} & R_{11} & R_{12} \\ \hline R_{20} & R_{21} & R_{22} \end{array}\right)$,
        $\left(\begin{array}{c} C_L \\ \hline\hline C_R \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} C_0 & C_1 & C_2 \end{array}\right)$, and $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline\hline T_1 \\ \hline T_2 \end{array}\right)$
            where $A_{11}$ is $k \times k$, $C_1$ has $k$ columns and $T_1$ has $k$ rows

        $\left[\left(\frac{R_{11}}{C_1}\right), \eta, b_1\right] := \left[\left(\frac{\{0\backslash R\}_{11}}{Y_1}\right), \eta, b_1\right] = QR\left(\left(\frac{R_{11}}{C_1}\right)\right)$
        Compute $T_1$ from $\left[\left(\frac{I}{Y_1}\right), \eta, b_1\right]$
        $\left(\frac{R_{12}}{C_2}\right) := \left(I + \left(\frac{I}{Y_1}\right) T_1^T \left(\begin{array}{c|c} I & Y_1^T \end{array}\right)\right) \left(\frac{R_{12}}{C_2}\right)$

    Continue with
        $\left(\begin{array}{c||c} R_{TL} & R_{TR} \\ \hline\hline R_{BL} & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} R_{00} & R_{01} & R_{02} \\ \hline R_{10} & R_{11} & R_{12} \\ \hline\hline R_{20} & R_{21} & R_{22} \end{array}\right)$,
        $\left(\begin{array}{c||c} C_L & C_R \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} C_0 & C_1 & C_2 \end{array}\right)$, and $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline\hline T_2 \end{array}\right)$
enddo
```

Figure 5: Blocked update to a QR factorization.

Note 3 *Notice that the algorithm is explicitly designed to take advantage of the zeros below the diagonal of R. As a result, factoring A followed by an update of the factorization requires essentially no more computation than the factorization in (1). Also, the Householder vectors that are stored below the diagonal are not overwritten. It is the case that an additional vector b is required to store the "β"s for the unblocked algorithm and an additional matrix is required to store the triangular "T" matrices for the blocked algorithm.*

4.2 Solving multiple Linear Least-Squares problems

If we now wish to compute multiple Linear Least-Squares solutions, one for each of the systems of linear equations defined by picking one column of the right-hand-side of

$$\left(\begin{array}{c} A \\ C \end{array}\right) X = \left(\begin{array}{c} B \\ D \end{array}\right)$$

the following approach will yield the desired result:

- Append $\left(\begin{array}{c|c} A & B \\ C & D \end{array}\right)$.
- Overwrite A with its factors $\{Y\backslash R\}$, also computing matrix T, as in Figure 2.
- Overwrite $\left(\begin{array}{c} R \\ C \end{array}\right)$ with $\left(\begin{array}{c} \bar{R} \\ Y^{(C)} \end{array}\right)$, also computing $T^{(C)}$ as in Figure 5.

```
Partition $B \rightarrow \left(\frac{B_T}{B_B}\right)$, $C \rightarrow \left( C_L \,\|\, C_R \right)$, and $T \rightarrow \left(\frac{T_T}{T_B}\right)$
    where  $C_L$ has 0 columns, and $B_T$ and $T_T$ have 0 rows

while $n(C_R) \neq 0$ do
    Determine block size $k$
    Repartition
        $\left( C_L \,\|\, C_R \right) \rightarrow \left( C_0 \,\|\, C_1 \,|\, C_2 \right)$,
        $\left(\frac{B_T}{B_B}\right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline\hline B_1 \\ \hline B_2 \end{array}\right)$, and $\left(\frac{T_T}{T_B}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline\hline T_1 \\ \hline T_2 \end{array}\right)$
            where  $C_1$ has $k$ columns and $B_1$ and $T_1$ have $k$ rows
    ----------------------------------------------------------------
    $\left(\frac{B_1}{D}\right) := \left( I + \left(\frac{I}{C_1}\right) T_1^T \left( I \,|\, C_1^T \right) \right) \left(\frac{B_1}{D}\right)$
    ----------------------------------------------------------------
    Continue with
        $\left( C_L \,\|\, C_R \right) \leftarrow \left( C_0 \,|\, C_1 \,\|\, C_2 \right)$,
        $\left(\frac{B_T}{B_B}\right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline B_1 \\ \hline\hline B_2 \end{array}\right)$, and $\left(\frac{T_T}{T_B}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline\hline T_2 \end{array}\right)$
enddo
```

Figure 6: Forward substitution consistent with the QR factorization of an updated matrix.

- Update B by forward substitution as in Figure 3.
- Update $\begin{pmatrix} B \\ D \end{pmatrix}$ by forward substitution using the Householder transformations computed as part of the update of R, as in Figure 6.
- Solve $\bar{R}X = B_T$ where B_T denotes the top n rows of the updated matrix B.

5 Out-of-Core Algorithms

Having now described the in-core algorithm, we can apply a similar strategy for problems that are too large to fit in the available memory of the machine. To deal with these problems, we have developed an out-of-core algorithm that allows us to store the bulk of the matrix components on disk, while only working on select pieces in-core at any one time. The algorithm we will outline here is unique in that it is both scalable and efficient.

5.1 Out-of-core QR factorization

Traditional OOC algorithms of the QR factorization have used a slab approach, in which the OOC matrix is processed by bringing into memory one or more slabs (blocks of columns) of the matrix at a time [6, 21, 7, 31, 27]. The problem with this technique is that it is inherently not scalable in the following sense: As the row dimension, m, of A becomes larger and larger, the width of the slab you can bring into memory becomes proportionally smaller. As m reaches into the millions, the number of columns able to be brought into memory numbers only in the dozens, even on today's powerful machines with large memories.

The alternative that we have found to the slab approach is to work with the matrix as a collection of tiles, where a tile is a submatrix that is roughly square. As was shown for the OOC Cholesky factorization [31, 26, 25, 17], a tiled approach provides true scalability. We will see that the processing of these tiles becomes a