CholeskyQR2: A Simple and Communication-Avoiding Algorithm for Computing a Tall-Skinny QR Factorization on a Large-Scale Parallel System

Takeshi Fukaya, Yuji Nakatsukasa, Yuka Yanagisawa and Yusaku Yamamoto

RIKEN Advanced Institute for Computational Science, Kobe, Japan
yuka.yanagisawa@riken.jp
The University of Tokyo, Tokyo, Japan
Waseda University, Tokyo, Japan
The University of Electro-Communications, Tokyo, Japan
JST CREST, Tokyo, Japan

Abstract—Designing communication-avoiding algorithms is crucial for high performance computing on a large-scale parallel system. The TSQR algorithm is a communication-avoiding algorithm for computing a tall-skinny QR factorization, and TSQR is known to be much faster and as stable as the classical Householder QR algorithm. The Cholesky QR algorithm is another very simple and fast communication-avoiding algorithm, but rarely used in practice because of its numerical instability. Our recent work points out that an algorithm that simply repeats Cholesky QR twice, which we call CholeskyQR2, gives excellent accuracy for a wide range of matrices arising in practice. Although the communication cost of CholeskyQR2 is twice that of TSQR, it has an advantage that its reduction operation is addition whereas that of TSQR is a QR factorization, whose high-performance implementation is more difficult. Thus, CholeskyQR2 can potentially be significantly faster than TSQR. Indeed, in our experiments using 16384 nodes of the K computer, CholeskyQR2 ran about three times faster than TSQR for a 4194304×64 matrix.

I. INTRODUCTION

It is now a consensus that communication-avoiding algorithms play a crucial role in high performance computing on a large-scale parallel system. Here, communication refers to data transfers among distributed processors over a network. The cost for communication is increasing relative to the cost for floating-point operations. Besides, the setup cost of communication (i.e. latency) becomes much larger than the cost for transferring a word of data (i.e. the inverse of bandwidth). Thus, it is vital to design an algorithm whose total number of data transfers (in other words, the number of messages) is much less than that in conventional algorithms, a so-called communication-avoiding algorithm [1].

The TSQR algorithm [2], [3] is a communication-avoiding algorithm for computing a tall-skinny QR factorization. A tall-skinny QR factorization, the QR factorization of a matrix with many more rows than columns, is a fundamental matrix factorization that appears in various numerical methods such as Krylov subspace methods [4], [5] and other subspace projection methods [6] for linear systems and eigenvalue problems. The widely-used Householder QR algorithm [7] requires very high communication cost, which makes it difficult to obtain good performance on a modern large-scale parallel system. TSQR is thus currently attracting much interest of many researchers.

On the other hand, the Cholesky QR algorithm [7, Thm. 5.2.3] is another communication-avoiding algorithm whose communication cost is equivalent to that of the TSQR algorithm (see Section II-B). Cholesky QR has the advantage over TSQR that its arithmetic cost is about half and that its reduction operator is addition, while that of TSQR is a QR factorization of a small matrix (for details see Section II-A). As a result, Cholesky QR usually runs faster than TSQR. However, Cholesky QR is rarely used in practice because of its instability: the distance from orthogonality of its computed Q grows rapidly with the condition number of the input matrix. By contrast, TSQR is unconditionally stable [2], [3], [8].

Recently [9], the authors have pointed out that the instability of Cholesky QR can be remedied simply by repeating the process twice: an algorithm that we refer to as CholeskyQR2. Specifically, CholeskyQR2 produces results as accurate as TSQR as long as the condition number is smaller than a threshold around $10^8$ (in double precision), indicating its practicality. Clearly, CholeskyQR2 requires twice as much communication as Cholesky QR, in terms of both the number of messages and the amount of transferred data. Still, it is much lower than that of Householder QR. When compared with TSQR, the communication cost of CholeskyQR2 is roughly twice that of TSQR, but it has an important advantage in the reduction operator over TSQR. Thus, CholeskyQR2 has a potential to be competitive with or outperform TSQR.

In this paper, we compare these two communication-avoiding algorithms for tall-skinny QR factorization: TSQR and CholeskyQR2. We illustrate the stability improvement of CholeskyQR2 with experimental results. We measure their parallel performance using the K computer and examine the difference in runtime. The main goal of the paper is to demonstrate the practicality of CholeskyQR2 on a modern...
large-scale parallel system.

The rest of the paper is organized as follows: in Section II, we briefly review TSQR and Cholesky QR. We then introduce CholeskyQR2 in Section III. We show experimental results both on the numerical stability and the parallel performance in Section IV, and discuss the results in Section V.

Throughout the paper, we let \( A \) be an \( m \times n \) real matrix with \( m \gg n \) and consider computing its QR factorization \( A = QR \), where \( Q \) is \( m \times n \) with orthonormal columns \( Q^T Q = I \) and \( R \) is \( n \times n \) upper triangular. We denote the condition number of \( A \) by \( \kappa(A) \) defined by \( \kappa(A) = \sigma_{\text{max}}(A) / \sigma_{\text{min}}(A) \), where \( \sigma_{\text{max}}(A) \) and \( \sigma_{\text{min}}(A) \) mean the maximum and minimum singular value of \( A \) respectively. We suppose that this computation is done by \( P \) distributed processors and make the following assumptions:

- \( P \) is a power of two, and \( m \) is divisible by \( P \) for simplicity.
- \( \frac{m}{P} \geq n \), since \( A \) is tall-skinny.
- \( A \) is stored in one-dimensional block row layout: \( A = [A_1^T \ A_2^T \ \cdots \ A_n^T]^T \), where \( A_i \) is \( \frac{m}{P} \times n \), and \( Q \) is stored in the same manner.
- The \( i \)-th processor has the data of \( A_i \) on entry and \( Q_i \) on exit. (No restriction on \( R \).)

II. REVIEW OF TSQR AND CHOLESKY QR

In this section, we review two communication-avoiding algorithms, namely the TSQR algorithm and the Cholesky QR algorithm, for computing a tall-skinny QR factorization.

A. The TSQR algorithm

The TSQR algorithm is an algorithm based on the idea that the QR factorization of \( A \) can be given by

\[
A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} \tilde{Q}_1 \tilde{R}_1 \\ \tilde{Q}_2 \tilde{R}_2 \end{bmatrix} = \begin{bmatrix} \tilde{Q}_1 & O \\ O & \tilde{Q}_2 \end{bmatrix} \tilde{Q}_{12} R, \tag{1}
\]

where

\[
\begin{bmatrix} \tilde{R}_1 \\ \tilde{R}_2 \end{bmatrix} = \tilde{Q}_{12} R. \tag{2}
\]

It is clear that (1) can be applied recursively to compute the QR factorization of \( A_1 \) (and \( A_2 \)). The QR factorization in (2) is a so-called structured QR factorization [3], which is essentially equivalent to the QR factorization of a \( 2n \times n \) matrix built by coupling two \( n \times n \) upper triangular matrices. By exploiting its triangular structure, one can compute it with much less arithmetic cost than the QR factorization of a general \( 2n \times n \) matrix. In the rest of the paper, we call the QR factorization of a general matrix a "general" QR factorization. For more detail on TSQR, see [2], [3].

Figure 1 illustrates the parallel computation of TSQR: computing \( R \) (Fig. 1(a)) and forming \( Q \) explicitly (Fig. 1(b)). Here we assume that a binary reduction tree is used (other reduction trees are described in [2], [3]). The process of merging the upper triangular matrices in computing \( R \) can be interpreted as a global reduction in which a structured QR factorization is the reduction operator. For forming \( Q \), the process can be regarded as one global broadcast that involves multiplications with a structured \( Q \) factor. In total, just two global collective communication operations is needed; much lower than that of the Householder QR algorithm, which requires \( O(n) \) global collective communication.

B. The Cholesky QR algorithm

The Cholesky QR algorithm is a simple algorithm, whose pseudocode is described in Fig. 2. One first calculates the Gram matrix \( A^T A \) by matrix-matrix multiplication, which is a great advantage in high performance computing. After that, one obtains \( R \) via the Cholesky factorization of the Gram matrix, and then compute \( Q \) via back substitution (e.g. the trsm routine in LAPACK), which is partly performed by matrix-matrix multiplication. It is worth noting that Cholesky QR belongs to the family of “triangular orthogonalization” type algorithms [10] (see line 3 in Fig. 2), in which \( Q \) is obtained by right-multiplying triangular matrices. This family includes the Gram-Schmidt type algorithms. The other family for QR factorization is “orthogonal triangularization”, which includes Householder QR and TSQR.

In parallel computing, Cholesky QR requires only one global communication operation, namely the allreduce operation for calculating the Gram matrix \( W \). The number of words transferred is about \( \frac{1}{2} n^2 \), corresponding to the data of the upper triangular part of \( W \). This means that Cholesky QR has about...
The communication cost of CholeskyQR2 is higher than TSQR, in both the number of messages and the amount of transferred data. The arithmetic cost in the former becomes $3\log_2 n^2$, almost all of which can be done by dgemm. Structured QR factorizations are calculated by our self-coded routine with loop blocking in which no BLAS routine is used. This is because BLAS routines are not tuned for such small matrices appearing in structured QR factorizations. We also compute the compact WY representation from the result of a structured QR factorization, and calculate the multiplication of structured $Q$ by $dtrmm$.

B. Numerical stability

We evaluated the numerical stability of CholeskyQR2 in parallel execution by using the FX10 supercomputing system Oakleaf-FX. We generated test matrices with condition number $\sigma$ by $A := USV$, where $U$ is an $m \times n$ random orthogonal matrix, $V$ is $n \times n$ random orthogonal and $\Sigma = \text{diag}(1, \sigma^{-1}, \cdots, \sigma^{-1}, \sigma)$. The obtained orthogonality and residual when $P = 8192$ are shown in Figs. 4(a) and 4(b) respectively. They illustrate that parallel CholeskyQR2 is stable until $\kappa(A)$ becomes larger than the threshold around $10^8$. These results agree with our recent theoretical analysis [9].

We note that there are many applications in which a Gram-Schmidt algorithm without reorthogonalization is used, e.g. QR factorizations in the block Arnold method [17] and orthogonalization in the real space DFT calculation [18]. The above results indicate that CholeskyQR2 can be applicable to these computations.

C. Parallel performance on the K computer

We measured the parallel execution time of CholeskyQR2 and TSQR on the K computer, which consists of 88192 computational nodes; each node has one SPARC64 VIII fx processor (2.0GHz, 8cores) and 16GB memory (DDR3 SDRAM, 64GB/s), and is connected by the 6D mesh/tours network (5GB/s/link, bidirectional). We compiled our codes written in Fortran90 by Fujitsu Fortran90 compiler with the option

---

Here, we regard all reduce operation as almost equal to one reduce and broadcast operations.

---

### III. THE CHOLESKYQR2 ALGORITHM

It has been reported that applying Cholesky QR repeatedly often produce a computed QR factorization with good numerical stability [11]. Recalling the stability improvement of the iterative Gram-Schmidt algorithms (e.g. CGS2 [12]), this idea is natural in view of the fact that Cholesky QR is a triangular orthogonalization algorithm. In this paper, we particularly focus on an algorithm that simply repeats Cholesky QR twice, which can be interpreted as a variant of the Cholesky QR algorithm with (one) reorthogonalization. We call this algorithm the CholksyQR2 algorithm, following the naming of the CGS2 algorithm. The pseudocode of CholeskyQR2 is described in Fig. 3.

In parallel computing, the communication cost of CholeskyQR2 is twice that of Cholesky QR, in both the number of messages and the amount of transferred data. The communication cost of CholeskyQR2 is higher than TSQR, but much less than the conventional Householder QR and Gram-Schmidt algorithms, which require $O(n)$ global collective communication operations. The costs of CholeskyQR2 in parallel execution are compared with those of TSQR in TABLE I, in which measurements are made along the critical path of the parallel execution.

### IV. EXPERIMENTAL RESULTS

We report the results of numerical experiments both on the numerical stability and on the parallel performance.

---

1$^1$Here, we regard all reduce operation as almost equal to one reduce and broadcast operations.

2The size of matrices in each dgemm call is of course not necessarily large.

3The system operated by SCD/ITC, The University of Tokyo, and it has almost the same architecture as the K computer.
TABLE I. PARALLEL EXECUTION COSTS OF THE TSQR AND CHOLESKYQR2 ALGORITHMS; COSTS FOR AN m × n MATRIX ARE MEASURED ALONG THE CRITICAL PATH OF THE PARALLEL EXECUTION WITH P PROCESSES.

<table>
<thead>
<tr>
<th>item</th>
<th>TSQR</th>
<th>CholeskyQR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#flops</td>
<td>$\frac{2}{3} n^2$ general QR factorization</td>
<td>$2 \times \frac{4}{3} n^2$ general matrix-matrix multiplication</td>
</tr>
<tr>
<td></td>
<td>$\frac{2}{3} n^2$ general Q factor multiplication</td>
<td>$2 \times \frac{4}{3} n^2$ back substitution</td>
</tr>
<tr>
<td></td>
<td>$\frac{4}{3} n^3 \log_2 P$ structured QR factorization</td>
<td>$2 \times \frac{4}{3} n^2 \log_2 P$ addition in reduce operation</td>
</tr>
<tr>
<td></td>
<td>$\frac{4}{3} n^3 \log_2 P$ structured Q factor multiplication</td>
<td>$\frac{4}{3} n^3$ triangular-triangular multiplication</td>
</tr>
</tbody>
</table>

| #msgs | $\log_2 P$ with structured QR factorizations | $2 \times \log_2 P$ in reduce operation |
|       | $\log_2 P$ with structured Q factor multiplications | $2 \times \log_2 P$ in broadcast operation |
| #words | $\frac{1}{2} n^2 \log_2 P$ with structured QR factorizations | $2 \times \frac{1}{2} n^2 \log_2 P$ in reduce operation |
|       | $\frac{1}{2} n^2 \log_2 P$ with structured Q factor multiplications | $2 \times \frac{1}{2} n^2 \log_2 P$ in broadcast operation |

![Comparison of the numerical stability: the orthogonality and residual of the computed QR factorizations by Householder QR (House), TSQR, Cholesky QR (Chol) and CholeskyQR2 (Chol2) are plotted.](image)

In this section, we discuss the parallel performance results shown in the previous section, particularly from the following three viewpoints.

A. Local kernel performance

Figs. 6(b) and 6(c) show that the performance gap in the local computational kernels (e.g. dgemm, recursive QR, etc., comparing those with the same color in each figure) strongly influences the difference of total execution time. To examine the generality of this gap, we measured the effective performance of each kernel not only on the K computer.
Fig. 5. Strong scaling on the K computer; the parallel execution times for computing the QR factorization of a $m \times n$ matrix by ScaLAPACK (pdgeqrf and pdorgqr), TSQR and CholeskyQR2 (Chol2) are plotted.
The arithmetic cost of structured QR factorization grows like $O(\log^2 P)$, this seems to be reasonable. Figs. 7(a) and 7(b) show the effective performance of each implementation of a structured QR factorization on the K computer and the Xeon processor. The figures corroborate that our implementation is appropriate and indicate that a high-performance implementation of a structured QR factorization is much more difficult than that of dgemm. In addition to the complications of computing a QR factorization, exploiting the upper triangular structure in structured QR factorizations makes the implementation more difficult. Tuning structured QR factorizations remains an vital issue for using TSQR more effectively.

\[ \text{TABLE II. PERFORMANCE GAP IN LOCAL COMPUTATIONAL KERNELS: THE RELATIVE EFFECTIVE RUNTIDES FOR ONE FLOATING-POINT OPERATION IN EACH KERNEL TO DGEHM ARE LISTED. } \]

\[ \frac{1}{P} = 2000 \text{ IN ALL CASES.} \]

\[ \begin{array}{cccccc}
 n & \text{dgemm} & \text{dsyrk} & \text{dtrmm} & \text{dtrsm} & \text{dgeqrf} & \text{rec_qr} \\
 16 & 1.00 & 1.75 & 0.67 & 0.79 & 4.11 & 4.83 \\
 32 & 1.00 & 4.20 & 1.32 & 1.55 & 12.8 & 9.74 \\
 64 & 1.00 & 4.21 & 1.62 & 1.80 & 23.9 & 10.8 \\
 128 & 1.00 & 3.16 & 1.99 & 2.19 & 37.5 & 9.76 \\
 256 & 1.00 & 1.93 & 2.14 & 2.23 & 16.9 & 6.42 \\
\end{array} \]

\[ \text{(a) Fujitsu BLAS, LAPACK on 1 node of the K computer (using 8 threads).} \]

\[ \begin{array}{cccccc}
 n & \text{dgemm} & \text{dsyrk} & \text{dtrmm} & \text{dtrsm} & \text{dgeqrf} & \text{rec_qr} \\
 16 & 1.00 & 1.52 & 1.58 & 1.00 & 8.01 & 7.61 \\
 32 & 1.00 & 1.16 & 1.43 & 0.85 & 5.48 & 6.54 \\
 64 & 1.00 & 1.38 & 1.27 & 1.02 & 4.37 & 6.25 \\
 128 & 1.00 & 1.43 & 1.21 & 1.19 & 2.65 & 4.16 \\
 256 & 1.00 & 1.29 & 1.13 & 1.10 & 2.43 & 2.92 \\
\end{array} \]

\[ \text{(b) Intel MKL ver. 11.0 on Xeon E5-2660 (using 8 threads).} \]

but also on a standard Xeon processor, and list the results in TABLEs II(a) and II(b) respectively. The result that the effective performance of dgeqrf and recursive QR are lower than that of the others is reasonable because the former is more sequential and complicated computations than the latter. This result reflects the advantage of CholeskyQR2 that it mainly consists of dgemm and dtrsm, which are expected to be highly tuned on a general system.

\[ \begin{array}{cccccc}
 n & \text{dgemm} & \text{dsyrk} & \text{dtrmm} & \text{dtrsm} & \text{dgeqrf} & \text{rec_qr} \\
 16 & 1.00 & 1.75 & 0.67 & 0.79 & 4.11 & 4.83 \\
 32 & 1.00 & 4.20 & 1.32 & 1.55 & 12.8 & 9.74 \\
 64 & 1.00 & 4.21 & 1.62 & 1.80 & 23.9 & 10.8 \\
 128 & 1.00 & 3.16 & 1.99 & 2.19 & 37.5 & 9.76 \\
 256 & 1.00 & 1.93 & 2.14 & 2.23 & 16.9 & 6.42 \\
\end{array} \]

\[ \text{(a) Fujitsu BLAS, LAPACK on 1 node of the K computer (using 8 threads).} \]

\[ \begin{array}{cccccc}
 n & \text{dgemm} & \text{dsyrk} & \text{dtrmm} & \text{dtrsm} & \text{dgeqrf} & \text{rec_qr} \\
 16 & 1.00 & 1.52 & 1.58 & 1.00 & 8.01 & 7.61 \\
 32 & 1.00 & 1.16 & 1.43 & 0.85 & 5.48 & 6.54 \\
 64 & 1.00 & 1.38 & 1.27 & 1.02 & 4.37 & 6.25 \\
 128 & 1.00 & 1.43 & 1.21 & 1.19 & 2.65 & 4.16 \\
 256 & 1.00 & 1.29 & 1.13 & 1.10 & 2.43 & 2.92 \\
\end{array} \]

\[ \text{(b) Intel MKL ver. 11.0 on Xeon E5-2660 (using 8 threads).} \]

\[ \text{TABLE II. PERFORMANCE GAP IN LOCAL COMPUTATIONAL KERNELS: THE RELATIVE EFFECTIVE RUNTIDES FOR ONE FLOATING-POINT OPERATION IN EACH KERNEL TO DGEHM ARE LISTED. } \]

\[ \frac{1}{P} = 2000 \text{ IN ALL CASES.} \]

\[ \begin{array}{cccccc}
 n & \text{dgemm} & \text{dsyrk} & \text{dtrmm} & \text{dtrsm} & \text{dgeqrf} & \text{rec_qr} \\
 16 & 1.00 & 1.75 & 0.67 & 0.79 & 4.11 & 4.83 \\
 32 & 1.00 & 4.20 & 1.32 & 1.55 & 12.8 & 9.74 \\
 64 & 1.00 & 4.21 & 1.62 & 1.80 & 23.9 & 10.8 \\
 128 & 1.00 & 3.16 & 1.99 & 2.19 & 37.5 & 9.76 \\
 256 & 1.00 & 1.93 & 2.14 & 2.23 & 16.9 & 6.42 \\
\end{array} \]

\[ \text{(a) Fujitsu BLAS, LAPACK on 1 node of the K computer (using 8 threads).} \]

\[ \begin{array}{cccccc}
 n & \text{dgemm} & \text{dsyrk} & \text{dtrmm} & \text{dtrsm} & \text{dgeqrf} & \text{rec_qr} \\
 16 & 1.00 & 1.52 & 1.58 & 1.00 & 8.01 & 7.61 \\
 32 & 1.00 & 1.16 & 1.43 & 0.85 & 5.48 & 6.54 \\
 64 & 1.00 & 1.38 & 1.27 & 1.02 & 4.37 & 6.25 \\
 128 & 1.00 & 1.43 & 1.21 & 1.19 & 2.65 & 4.16 \\
 256 & 1.00 & 1.29 & 1.13 & 1.10 & 2.43 & 2.92 \\
\end{array} \]

\[ \text{(b) Intel MKL ver. 11.0 on Xeon E5-2660 (using 8 threads).} \]
practicality of CholeskyQR2, which is strengthened also by the simplicity of its implementation.

Many issues remain to be solved for an effective use of CholeskyQR2. First, a thorough performance evaluation on systems other than the K computer is necessary. Moreover, in addition to the binary tree based TSQR algorithm, we need to compare CholeskyQR2 with other state-of-the-art communication-avoiding algorithms mentioned in Section V-C. The numerical stability of CholeskyQR2 also needs to be examined with matrices appearing in practical computations.

The second issue pertains to the numerical breakdown in the Cholesky factorization. Throughout our experiments we observe that CholeskyQR2 gives an accurate result if the first Cholesky factorization does not break down, but a rigorous analysis is unavailable. One possible approach is to first try a Cholesky factorization and then determine whether to continue CholeskyQR2 or switch to TSQR (if Cholesky breaks down). This seems to be reasonable because the experimental results indicate that the additional cost entailed would not be so expensive relative to the cost of TSQR.

Third, dealing with cases other than tall-skinny matrices is also of importance. One may employ the panel blocking technique as in block Gram-Schmidt [22] and use CholeskyQR2 for computing the QR factorization of a panel. Additionally, it is worth considering the case when a computing a representation of the orthogonal transformation instead of the explicit $Q$, such as in a symmetric-band transformation for symmetric eigenvalue problems, one can construct the compact-WY representation from the explicit $Q$ computed by CholeskyQR2 as in TSQR [23]. For both cases, a theoretical error analysis is not yet available.

**ACKNOWLEDGMENT**

The authors thank the anonymous referees for their valuable comments. We also thank Dr. Toshiyuki Imamura for his useful advice and Dr. Keita Teranishi for his suggestion of submitting our work to ScalA2014. The first author appreciates the fruitful discussions during his recent visit to LBNL and UC Berkeley. This research was supported by JST, CREST and used computational resources of the K computer provided by the RIKEN AICS through the HPCI System Research project (Project ID:hp120170).

**REFERENCES**


