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

METR 2014–37 December 2014

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*
takeshi.fukaya@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.*

1. 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 2.2). 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 2.1). As a result, Cholesky QR usually runs faster than TSQR. However, Cholesky QR is rarely used in practice because of its instability: the orthogonality of its computed Q grows rapidly with the condition number $\kappa(A) = \|A\|_2\|A^{-1}\|_2$ of the input matrix. By constrast, 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 $\kappa(A)$ is smaller than a threshold around 10^8 (in double precision), indicating its practicality. Cleary, 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 ChoeleskyQR2 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 2, we briefly review TSQR and Cholesky QR. We then introduce CholeskyQR2 in Section 3. We show experimental results both on the numerical stability and the parallel performance in Section 4, and discuss the results in Section 5.

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 $Q^TQ = I$ and R is $n \times n$ upper triangular. 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; $A = [A_1^\top A_2^\top \cdots A_P^\top]^\top$, where A_i is $\frac{m}{P} \times n$, and Q is stored in the same manner.
- The ith processor has the data of A_i on entry and Q_i on exit. (No restriction on R.)

2. Review of TSQR and Cholesky QR

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

2.1. The TSQR algorithm

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

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} \tilde{Q}_1\tilde{R}_1 \\ \tilde{Q}_2\tilde{R}_2 \end{bmatrix} = \left(\begin{bmatrix} \tilde{Q}_1 & O \\ O & \tilde{Q}_2 \end{bmatrix} \tilde{Q}_{12} \right) R, \quad (1)$$

where

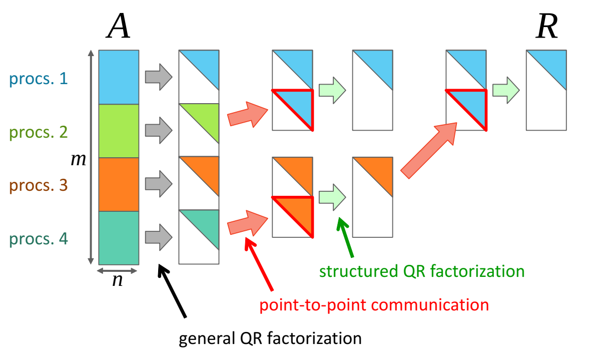
$$\begin{bmatrix} \tilde{R}_1 \\ \tilde{R}_2 \end{bmatrix} = \tilde{Q}_{12} R. \quad (2)$$

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].

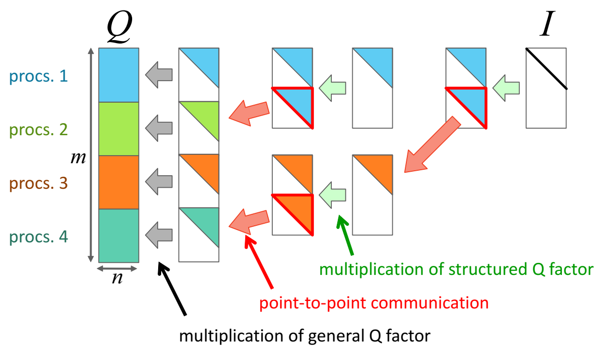
Figures 1 illustrates the parallel computation of TSQR: computing R (Figure 1(a)) and forming Q explicitly (Figure 1(b)). Here we assume that a binary reduction tree is used (other reduction trees are described in [2], [3]). The process of merging the upper triangular matrices in computing R can be interpreted 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.

2.2. The Cholesky QR algorithm

The Cholesky QR algorithm is a simple algorithm, whose pseducode is described in Figure 2. One first calculates the Gram matrix $A^\top A$ 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 Figure 2), in which Q is obtained by right-multiplying triangular matrices. This family includes the Gram-Schmidt type algorithms. The other family for QR



(a) Computation of R


Q
I
procs. 1
procs. 2
procs. 3
procs. 4
m
n
multiplication of structured Q factor
point-to-point communication
multiplication of general Q factor

(b) Generation of explicit Q

Figure 1. Sketch of the parallel computation of the binary tree based TSQR algorithm when $P = 4$.

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 same communication cost as TSQR[1].

However, it is well known that Cholesky QR is numerically unstable because the Gram matrix $A^\top A$ squares the condition number of A while TSQR is as stable as Householder QR [3], [8]. The loss of orthogonality of the computed Q by the Cholesky QR algorithm grows with $\kappa(A)$ like $O(\epsilon\kappa(A)^2)$ where $\epsilon \approx 10^{-16}$.

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

```
Input: $A$; $A$ is $m \times n$
  1: $W := A^\top A$
  2: $R :=$ Chol($W$) // Cholesky factorization
  3: $Q := AR^{-1}$
Output: $Q, R$; $Q$ is $m \times n$ orthonormal, $R$ is $n \times n$
    upper triangular, $QR = A$
```

Figure 2. Pseudocode of the Cholesky QR algorithm.

```
Input: $A$; $A$ is $m \times n$
  1: $[A', R_1] :=$ CholeskyQR($A$)
  2: $[Q, R_2] :=$ CholeskyQR($A'$)
  3: $R := R_2 R_1$
Output: $Q, R$; $Q$ is $m \times n$ orthonormal, $R$ is $n \times n$
    upper triangular, $QR = A$
```

Figure 3. Pseudocode of the CholeskyQR2 algorithm.

3. 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 CholskyQR2 algorithm, following the naming of the CGS2 algorithm. The pseudocode of CholeskyQR2 is described in Figure 3.

In parallel computing, the communication cost of CholeskyQR2 is twice that of Choleksy 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 Gran-Schmidt algorithms, which require $O(n)$ global collective communication operations. The costs of CholeksyQR2 in parallel execution are compared with those of TSQR in Table 1, in which measurements are made along the critical path of the parallel execution.

4. Experimental results

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

Table 1. Parallel execution costs of the TSQR and CholeskyQR2 algorithms; costs for an $m \times n$ matrix are measured along the critical path of the parallel execution with P processes.

item	TSQR		CholeskyQR2	
#flops	$2\frac{m}{P}n^2$	general QR factorization	$2 \times \frac{m}{P}n^2$	general matrix-matrix multiplication
	$2\frac{m}{P}n^2$	general Q factor multiplication	$2 \times \frac{m}{P}n^2$	back substitution
			$2 \times \frac{1}{3}n^3$	Cholesky factorization
			$\frac{1}{3}n^3$	triangular-triangular multiplication
	$\frac{2}{3}n^3 \log_2 P$	structured QR factorization	$2 \times \frac{1}{2}n^2 \log_2 P$	addition in reduce operation
	$\frac{2}{3}n^3 \log_2 P$	structured Q factor multiplication		
#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 multiplications	$2 \times \frac{1}{2}n^2 \log_2 P$	in broadcast operation

4.1. Implementation

4.1.1. CholeskyQR2. Our CholeskyQR2 code is implemented based on the routines provided in the BLAS and LAPACK libraries. There is some performance gap between dgemm and dsyrk for computing $A^\top A$ (Table 3a), and we employed dgemm although the number of the floating-point operations becomes double. We use dpotrf for computing the Cholesky factorization, dtrsm for the back substitution and dtrmm for the products of R_1 and R_2, where zeros are inserted into the strictly lower triangular part of R_1. We transfer the whole data of $A^\top A$ to avoid the cost for copying the upper triangular part to/from a continuous buffer from/to the general rectangular array required in dgemm etc.

4.1.2. TSQR. Our TSQR code is implemented as follows. We adopt the recursive QR algorithm [13], known to be efficient particularly in computing the full compact WY representation [14], [15], for the general QR factorization in each process. Consequently, the multiplication of Q, whose arithmetic cost is reduced to about half the original amount [16], is done by one large dtrmm and other two small dtrmms. The arithmetic cost in the former becomes $3\frac{m}{P}n^2$, almost all of which can be done by dgemm[2]. Structured QR factorizations are calculated by our self-coded routine 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.

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

4.2. Numerical stability

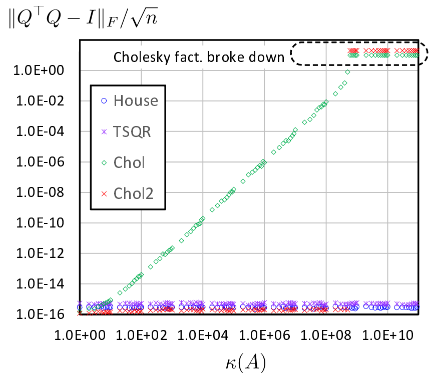
We evaluated the numerical stability of CholeskyQR2 in parallel execution by using the FX10 supercomputing system Oakleaf-FX[3]. We generated test matrices with condition number σ by $A := U\Sigma V$, where U is an $m \times n$ random orthogonal matrix, V is $n \times n$ random orthogonal and $\Sigma = \mathrm{diag}(1, \sigma^{\frac{1}{n-1}}, \cdots, \sigma^{\frac{n-2}{n-1}}, \sigma)$. The obtained orthogonality and residual when $P = 8192$ are shown in Figures 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 Arnolid method [17] and orthogonalization in the real space DFT calculation [18]. The above results indicate that CholeskyQR2 can be applicable to these computations.

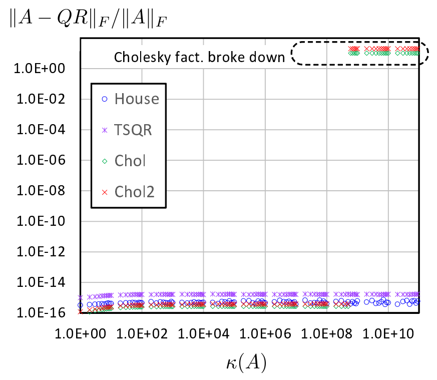
4.3. Parallel performance on the K computer

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 VIIIfx processor (2.0GHz, 8cores) and 16GB memory (DDR3 SDRAM, 64GB/s), and is connected by the 6D mesh/tours network (5GB/s/link,

3. the system operated by SCD/ITC, The University of Tokyo, and it has almost the same architecture as the K computer.



(a) Orthogonality: $\|Q^\top Q - I\|_F/\sqrt{n}$



(b) Residual: $\|A - QR\|_F/\|A\|_F$

Figure 4. 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. $m = 2097152$, $n = 64$ and $P = 8192$.

bidirectional). We complied our codes written in Fortran90 by Fujitsu Fortran90 compiler with the option "-Kfast,openmp" and linked Fujitsu BLAS, LAPACK and MPI libraries. Note that we used the thread parallelized BLAS and LAPACK libraries. The version of the language environment was K-1.2.0-15. We assigned one MPI process per node and eight threads per process. Since the condition number of A does not influence the execution time, we used a random matrix whose condition number is not so large.

Figures 5(a) to 5(f) show the strong scaling of ScaLAPACK (pdgeqrf and pdorgqr), TSQR and CholeskyQR2. These graphs show that communication-avoiding algorithms (TSQR and CholeskyQR2) are much faster than ScaLAPACK and that CholeskyQR2 is faster than TSQR in almost all cases. We also observe that the runtime difference between TSQR and CholeskyQR2 depends not on m but rather on n when P is large enough.

We then show the breakdown of the execution time of four typical cases in Figures 6(a) to 6(d). When P is small and n is also small, the performance gap between dgemm and recursive QR is not so large (the dark blue parts in Figure 6(a)), thus the total execution time of TSQR and that of CholeskyQR2 are almost the same. We suspect that this is because $\frac{m}{P} \gg n$, so each local matrix is still tall-skinny. However, as n grows, the performance gap also becomes large, which makes CholeskeyQR2 faster than TSQR (Figure 6(b)). On the other hand, when P is large, the cost of calculating the structured QR factorizations becomes a serious bottleneck in TSQR (the orange part in Figure 6(c)), and its influence grows as n becomes large (Figure 6(d)).

5. Discussion

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

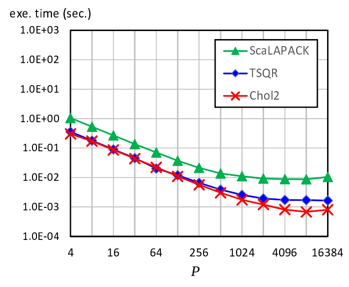
5.1. Local kernel performance

Figures 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 but also on a standard Xeon processor, and list the results in Tables 2(a) and 2(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.

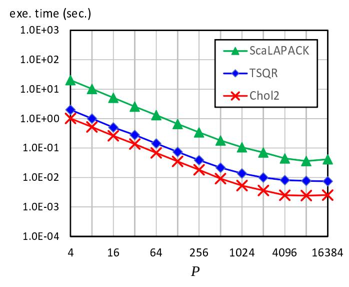
5.2. Performance of structured QR factorizations

Figures 6(c) and 6(d) clearly show that the runtime for computing structured QR factorizations becomes a serious bottleneck of TSQR when P is large. Taking into account that the arithmetic cost of structured QR factorization grows like $O(\log_2 P)$, this seems to be reasonable. Figures 7(a) and 7(b) show the effective performance of each implementation[4] of a structured

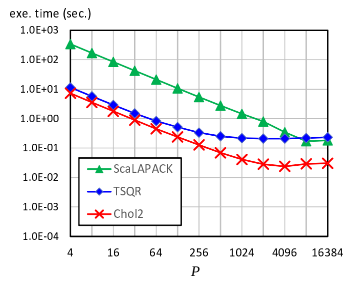
4. dtpqrt is the specialized routine in LAPACK for computing a structured QR factorization.



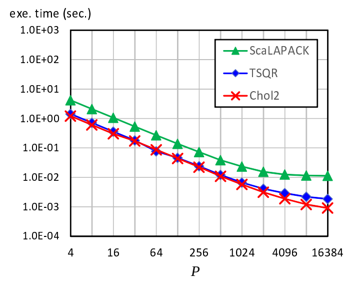
(a) $m = 4194304, n = 16$



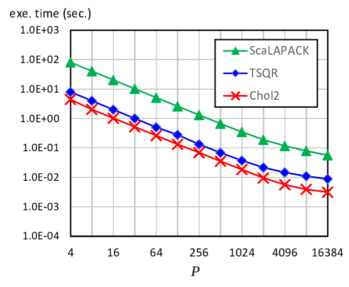
(b) $m = 4194304, n = 64$



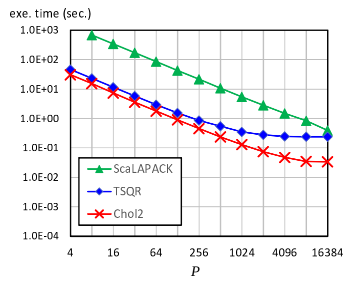
(c) $m = 4194304, n = 256$



(d) $m = 16777216, n = 16$



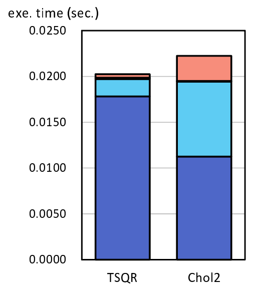
(e) $m = 16777216, n = 64$



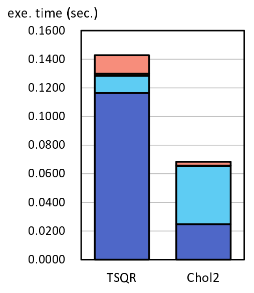
(f) $m = 16777216, n = 256$

Figure 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.

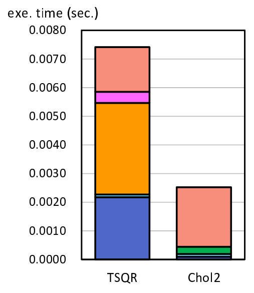
TSQR :	🟦	general QR fact. (recursive QR)	🟦	general Q mult. (dtrmm)	🟧	structured QR fact.	🟪	structured Q mult. (dtrmm)	🟥	comm.
Chol2 :	🟦	mat-mat mult. (dgemm)	🟦	back substitution (dtrsm)	🟩	Cholesky fact. (dpotrf) & tri-tri mult. (dtrmm)			🟥	comm.



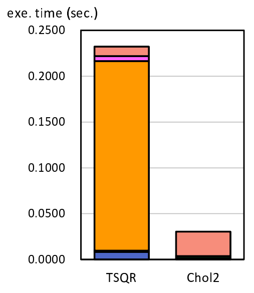
(a) $n = 16, P = 64$



(b) $n = 64, P = 64$



(c) $n = 64, P = 16384$



(d) $n = 256, P = 16384$

Figure 6. Breakdown of the execution time; typical breakdowns of the parallel execution time of TSQR and CholeskyQR2 (Chol2) are shown. $m = 4194304$ in all cases.

QR factorization on the K computer and the Xeon processor. The figures corroborate that our implementation is appropriate and indicate that the 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[5] in structured QR factorizations makes the implementation more difficult.

5. dgeqrf can not exploit the triangular structure. Its #flops is thus quintupled, and its effective performance is also not high.

5.3. Other state-of-the-art algorithms

The TSQR algorithm against which we compared CholeskyQR2 is a basic variant that uses a binary reduction tree, and there are some state-of-the-art communication-avoiding algorithms, e.g. the tile QR

Table 2. Performance gap in local computational kernels; the relative effective runtimes for one floating-point operation in each kernel to dgemm are listed. " req_qr" is our implementation of recursive QR. $\frac{m}{P} = 2000$ in all cases.

n	dgemm	dsyrk	dtrmm	dtrsm	dgeqrf	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

(a) Fujitsu BLAS, LAPACK on 1 node of the K computer (using 8 threads).

n	dgemm	dsyrk	dtrmm	dtrsm	dgeqrf	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

(b) Intel MKL ver. 11.0 on Xeon E5-2660 (using 8 threads).

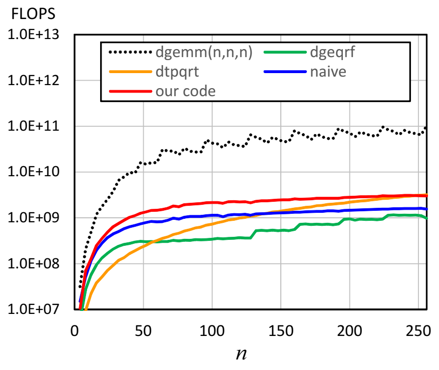
algorithm [19]. For tall and skinny matrices, a binary reduction tree is known to be near optimal. As n becomes large, other trees such as hierarchical trees [20] are probably more efficient, e.g. the case of $n = 256$; our experiments simply show that CholeskyQR2 is faster than a basic TSQR algorithm, which may be suboptimal when n is not small.

We also note that applying the TSQR algorithm for computing local (in node) QR factorizations sometimes improves the performance [21], which reduces the gap in the local kernel performance as discussed in Section 5.1.

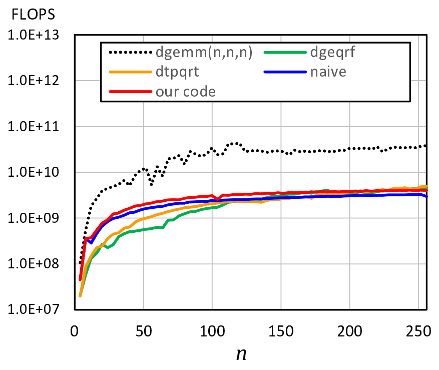
6. Conclusion

Our main message is that CholeskyQR2 is a practical algorithm worth considering for computing a tall-skinny QR factorization on a modern large-scale parallel system. We confirmed that it is as stable as the TSQR algorithm for a wide range of matrices arising in practice. We demonstrated that it can significantly outperform the binary tree based TSQR algorithm on the K computer. The results suggest the overall practicality of CholeskyQR2, which is strengthened also by the simplicity of its implementation.

Many issues remain to be solved for an effective use



(a) Fujitsu BLAS, LAPACK on 1 node of the K computer (using 8 threads).



(b) Intel MKL ver. 11.0 on Xeon E5-2660 (using 8 threads).

Figure 7. Effective performance of routines that compute a structured QR factorization.

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 5.3. 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.

Acknowledgment

The authors would like to thank Dr. Toshiyuki Imamura for his valuable comments. 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

[1] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, “Minimizing communication in numerical linear algebra,” *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 3, pp. 866–901, 2011.

[2] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, “Communication-optimal parallel and sequential QR and LU factorizations,” *SIAM J. Sci. Comp*, vol. 34, no. 1, pp. 206–239, 2012.

[3] ——, “Communication-avoiding parallel and sequential QR factorizations,” *CoRR*, vol. abs/0806.2159, 2008.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Philadelphia, PA, USA: SIAM, 2000.

[5] M. H. Gutknecht, “Block Krylov space methods for linear systems with multiple right-hand sides: An introduction,” 2006.

[6] T. Sakurai and H. Sugiura, “A projection method for generalized eigenvalue problems using numerical integration,” *J. Comput. Appl. Math.*, vol. 159, no. 1, pp. 119–128, Oct. 2003.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. The Johns Hopkins University Press, 2012.

[8] D. Mori, Y. Yamamoto, and Z. Shao-Liang, “Backward error analysis of the AllReduce algorithm for Householder QR decomposition,” *Jpn J. Ind. Appl. Math.*, vol. 29, no. 1, pp. 111–130, feb 2012.

[9] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya, “Roundoff error analysis of the CholeskyQR2 algorithm,” http://www.na.scitec.kobe-u.ac.jp/˜yamamoto/work/CholeskyQR2.pdf, submitted.

[10] L. N. Trefethen and I. David Bau, *Numerical Liner Algebra*. Philadelphia: SIAM, 1997.

[11] A. Stathopoulos and K. Wu, “A block orthogonalization procedure with constant synchronization requirements,” *SIAM J. Sci. Comp*, vol. 23, pp. 2165–2182, 2002.

[12] N. N. Abdelmalek, “Round off error analysis for Gram-Schmidt method and solution of linear least squares problems,” *BIT*, vol. 11, 1971.

[13] E. Elmroth and F. G. Gustavson, “Applying recursion to serial and parallel QR factorization leads to better performance,” *IBM J. RES. DEV.*, vol. 44, no. 4, pp. 605–624, 2000.

[14] C. Puglisi, “Modification of the Householder method based on the compact WY representation,” *SIAM J. Sci. Stat. Comp.*, vol. 13, pp. 723–726, 1992.

[15] R. Schreiber and C. F. van Loan, “A storage-efficient WY representation for products of Householder transformations,” *SIAM J. Sci. Stat. Comp.*, vol. 10, pp. 53–57, 1989.

[16] Y. Yamamoto, T. Fukaya, T. Uneyama, M. Takata, K. Kimura, M. Iwasaki, and Y. Nakamura, “Accelerating the singular value decomposition of rectangular matrices with the CSX600 and the Integrable SVD,” in *Lecture Notes in Computer Science*, vol. 4671, 2007, pp. 340–345.

[17] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*. Manchester, UK: Manchester University Press, 1992.

[18] Y. Hasegawa, J.-I. Iwata, M. Tsuji, D. Takahashi, A. Oshiyama, K. Minami, T. Boku, F. Shoji, A. Uno, M. Kurokawa, H. Inoue, I. Miyoshi, and M. Yokokawa, “First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the k computer,” in *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC’11)*, 2011, pp. 1–11.

[19] F. Song, H. Ltaief, B. Hadri, and J. Dongarra, “Scalable tile communication-avoiding QR factorization on multicore cluster systems,” in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC’10)*, 2010, pp. 1–11.

[20] J. Dongarra, M. Faverge, T. HéRault, M. Jacquelin, J. Langou, and Y. Robert, “Hierarchical QR factorization algorithms for multi-core clusters,” *Parallel Comput.*, vol. 39, no. 4-5, pp. 212–232, Apr. 2013.

[21] M. Hoemmen, “A communication-avoiding, hybrid-parallel, rank-revealing orthogonalization method.” in *IPDPS*. IEEE, 2011, pp. 966–977.

[22] G. Stewart, “Block Gram–Schmidt orthogonalization,” *SIAM J. Sci. Comp*, vol. 31, pp. 761 – 775, 2008.