Performance Analysis of the Householder-type Parallel Tall-Skinny QR Factorizations toward Automatic Algorithm Selection

Takeshi Fukaya[1,3], Toshiyuki Imamura[1,3] and Yusaku Yamamoto[2,3]

[1] RIKEN Advanced Institute for Computational Science, Kobe, Japan
takeshi.fukaya@riken.jp
[2] The University of Electro-Communications, Tokyo, Japan
[3] JST CREST

Abstract. We consider computing tall-skinny QR factorizations on large-scale parallel machines and analyze the execution times of the Householder QR algorithm and the TSQR algorithm using a realistic performance model. We point out the main cause of the difference between the execution time of Householder QR and that of TSQR. We also discuss a way to estimate which algorithm is faster for a given condition. Numerical experiments on the K computer support our analyses and show our success in determining the faster algorithm.

1 Introduction

The QR factorization of a tall and skinny matrix, which has many more rows than columns, appears in many numerical computations. In block subspace projection methods for sparse linear systems and eigenvalue problems, an orthogonal basis of the subspace is often calculated via the tall-skinny QR factorization. As these methods are used on large-scale parallel machines, an efficient algorithm for computing tall-skinny QR factorizations on such machines is strongly required.

Because of the increasing costs of transferring messages between processors, referred to as *communication*, on today's parallel machines, so-called *communication-avoiding* algorithms have been extensively studied. The lower bound for communication costs of a QR factorization has been derived by Demmel et al. [1]. They have also presented an algorithm named TSQR for tall-skinny QR factorizations and shown that their algorithm attains the lower bound but the conventional Householder QR algorithm, commonly used due to its excellent numerical stability, does not. In addition, several cases where TSQR outperforms Householder QR are reported.

However, their main interest lies in showing the optimality of TSQR from the theoretical point of view, and not in predicting the performance of TSQR or Householder QR accurately. They therefore use a rather simple and abstract model that does not necessarily reflect the characteristic of actual machines. For example, they assume that the effective performance of a floating-point operation is independent of the computing kernel. Such a simple model would not be sufficient to predict the performance difference of TSQR and Householder QR for a given matrix size and a computational environment. In fact, in our evaluation of TSQR on the K computer, we have observed several

cases where TSQR is slower than Householder QR, which can hardly be expected from their performance model.

In this paper, we analyze the difference between the parallel execution time of TSQR and that of Householder QR by using a more realistic model. Our model uses different effective floating-point throughputs for different computational kernels. We also show how it improves the accuracy of the model to deviate from a simple linear model and allow the effective performance to depend on the data size. We aim to predict the difference based on our analysis, which realizes the automatic algorithm selection depending on a given condition. The results of the numerical experiments on the K computer were in general agreement with our analysis.

2 Tall-Skinny QR factorization

Let A be an $m \times n$ real matrix and $m \gg n$, meaning A is tall and skinny. The factorization $A = QR$, where Q is an $m \times n$ matrix with orthonormal columns and R is an $n \times n$ upper triangular matrix, is called the thin QR factorization of A [2]. In this paper, P distributed processors connected by a network are assumed to store A in a one-dimensional block row layout: $A = [A_1^\top \ A_2^\top \ \cdots \ A_P^\top]^\top$. Then, the QR factorization of A is considered to be calculated by a parallel algorithm. Note that m_i denotes the number of rows of A_i and that $m_i \geq n$ is assumed for the reason that A is tall and skinny.

In this paper, we discuss the two parallel algorithms of computing tall-skinny QR factorizations: the Householder QR algorithm, which has been used widely in practical computations, and the TSQR algorithm, which has been presented as a communication-avoiding algorithm by Demmel et al. [1].

2.1 The Householder QR algorithm

In the Householder QR algorithm, the target matrix A is transformed into the upper triangular matrix R by a sequence of the Householder transformations $H_i := I - t_i \boldsymbol{y}_i \boldsymbol{y}_i^\top \ (i = 1, \ldots, n)$, which implicitly represents Q. This algorithm consists of the iteration of the two steps: generation of the Householder transformation from the target column vector, and application of the Householder transformation: $H_i A = A - t_i \boldsymbol{y}_i (\boldsymbol{y}_i^\top A)$. For details of the Householder QR algorithm, see [2].

In parallel computing, where A is distributed as previously mentioned, at least two collective communications are required in the Householder QR algorithm: for calculating the 2-norms of column vectors and matrix-vector multiplications $\boldsymbol{y}_i^\top A$. Thus, $2n$ communications are totally necessary in the whole algorithm.

2.2 The TSQR algorithm

TSQR is an algorithm based on the idea that $A = QR$ can be given by

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} Q_1 R_1 \\ Q_2 R_2 \end{bmatrix} = \begin{bmatrix} Q1 & O \\ O & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \end{bmatrix} = \left(\begin{bmatrix} Q1 & O \\ O & Q_2 \end{bmatrix} Q_{12} \right) R,$$

where Q_1 and Q_2 are $\frac{m}{2} \times n$ orthogonal, R_1 and R_2 are $n \times n$ upper triangular and Q_{12} is $2n \times n$ orthogonal. It is clear that this idea is recursively applicable to the QR factorizations of A_1 and A_2 while $m \geq 2n$. In the TSQR algorithm, the QR factorizations of the partitioned matrices are first calculated, and the obtained R_i $(i = 1, \ldots, P)$ are then reduced into R by the QR factorizations of the matrices built by coupling two upper triangular matrices, which we call the *structured* QR factorization. Although there are several reduction trees to calculate R [1], we assume that P is a power of two and that a binary reduction tree is used.

In parallel computing, each processor first compute the QR factorization $A_i = Q_i R_i$, which we call the *generall* QR factorization. After that, one repeats sending/receiving one's R to/from one's neighbor by a point-to-point communication and calculating the structured QR factorization. Through this computation, only $\log_2 P$ point-to-point communications are required. On the other hand, structured QR factorizations can be cheaply calculated by exploiting the structure of the matrix. Therefore, TSQR is considered to be superior to Householder QR from the viewpoint of communication-avoiding.

3 Analysis of the execution times based on a realistic model

In this section, we present a more realistic performance model and analyze the difference between the parallel execution time of TSQR and that of Householder QR. We also discuss how to predict the difference toward automatic algorithm selection.

3.1 Performance modeling

Taking account of some aspects in practical computing, we modify the performance model which Demmel et al. used to discuss the communication optimality of the algorithms [1]. According to their model, parallel execution times are expressed as

$$T = \gamma \cdot (\#flops) + \alpha_{1to1} \cdot (\#msgs) + \beta_{1to1} \cdot (\#words), \tag{1}$$

where γ is the floating-point throughput, α_{1to1} and β_{1to1} are respectively the setup cost and the inverse of the network bandwidth in a point-to-point communication. However, this model ignores the practical aspect that γ often depends on the computing kernels, that is, the patterns of floating-point operations and memory access.

The computing kernel in Householder QR is easily verified to be almost equivalent to that of the general QR factorization in TSQR. On the other hand, that of the structured QR factorization in TSQR should be distinguished from them because both the size and structure of the matrix are obviously different from those in the general QR factorization. For this reason, we introduce γ_{ge} and γ_{st}, where γ_{ge} means the effective floating-point throughput in the general QR factorization and γ_{st} means that in the structured QR factorization. Hence, we model the execution times as

$$T = \gamma_{\text{ge}} \cdot (\#flops\_ge) + \gamma_{\text{st}} \cdot (\#flops\_st) + \alpha_{1to1} \cdot (\#msgs) + \beta_{1to1} \cdot (\#words). \tag{2}$$

We list the value of *#flops_ge*, *#flops_st*, *#msgs* and *#words* in each algorithm in Table 1, where we refer to [1].

Table 1. Performance models of the Householder QR algorithm and the TSQR algorithm:

Algorithm	$\#flops\_ge$	$\#flops\_st$	$\#msgs$	$\#words$
Householder	$2\frac{m}{P}n^2$	0	$2n \cdot \log_2 P$	$\frac{n^2}{2} \cdot \log_2 P$
TSQR	$2\frac{m}{P}n^2 - \frac{2}{3}n^3$	$\frac{2}{3}n^3 \cdot \log_2 P$	$\log_2 P$	$\frac{n^2}{2} \cdot \log_2 P$

3.2 Analysis

Using the modified model, we analyze the difference between the execution time of Householder QR and that of TSQR. From Table 1, the difference is given as

$$\begin{aligned} T_{\text{diff}} &:= T_{\text{Householder QR}} - T_{\text{TSQR}} \\ &= \gamma_{\text{ge}} \cdot \frac{2}{3}n^3 - \gamma_{\text{st}} \cdot \frac{2}{3}n^3 \log_2 P + \alpha_{\text{1to1}} \cdot (2n \log_2 P - \log_2 P) \\ &\simeq (\alpha_{\text{1to1}} \cdot 2n - \gamma_{\text{st}} \cdot \frac{2}{3}n^3) \cdot \log_2 P, \end{aligned} \tag{3}$$

here we assume that $\gamma_{\text{st}} \cdot \log_2 P \gg \gamma_{ge}$ and that $2n \gg 1$. The former one is based on the consensus that the high-performance implementation of a routine for a small and structured matrix is often much more difficult than that for a general and large matrix. Eq. (3) means that the difference between the execution time of Householder and that of TSQR is mainly caused by the difference between the setup costs of communications in Householder QR and the floating-point operation costs in the structured QR factorizations in TSQR.

This result indicates that $T_{\text{diff}} < 0$, meaning TSQR is slower than Householder QR, when $n > \sqrt{3\alpha_{\text{1to1}}/\gamma_{\text{st}}}$. Thus, users need to determine which algorithm is the faster in their target computational conditions. The result also shows that the determination of the faster algorithm depends not on m and P but only on n while the platform and the implementation are fixed.

3.3 Toward auto-tuning

We discuss the prediction of T_{diff}, which leads to the automatic algorithm selection. T_{diff} can be straightforwardly estimated by Eq. (3) because α_{1to1} and γ_{st} are easily measured by executing benchmark programs; the benchmark for measuring α_{1to1} requires only two processes and that for γ_{st} requires only one, and both consume only a little time.

More reliable estimations of T_{diff} are expected to be given by

$$T_{\text{diff}} \simeq \alpha_{\text{all}}(P) \cdot 2n - T_{\text{st}}(n) \cdot \log_2 P, \tag{4}$$

where $\alpha_{\text{all}}(P)$ is the setup cost of a collective communication (allreduce) with P processes and $T_{\text{st}}(n)$ is the execution time of the structured QR factorizations of a matrix which consists of two $n \times n$ upper triangular matrices. Although more time and processes are required to measure $\alpha_{\text{all}}(P)$ and $T_{\text{st}}(n)$, it would be acceptable in practical situations due to their reusability; $\alpha_{\text{all}}(P)$ can be used for any n, $T_{\text{st}}(n)$ for any P and both for any m.

4 Numerical experiments

4.1 Experimental conditions

We verified our analyses described in the previous section through numerical experiments on the K computer, which consists of 88129 computational nodes; each node has one SPARC64 VIIIfx processor (2.0GHz, 8 cores) and 16GB memory, and is connected by the 6D mesh/torus network (5GB/sec, per link, bidirectional).

We implemented a parallel Householder QR program (denoted *House* in the followings) and two kinds of parallel TSQR programs (*TSQR 1* / *TSQR 2*); the structured QR factorization in TSQR 1 was implemented with BLAS routines, and that in TSQR 2 was done in a simple way with the loop blocking technique. The general QR factorization implemented with BLAS routines was employed in the both TSQR programs. Note that we used the MPI and BLAS libraries provided from Fujitsu on the K computer, assigned one MPI process per one node, and used parallel version of the BLAS routines (8 threads/node). We show the performance of our two structured QR programs and general QR program in Fig. 1(a) and the setup costs of MPI routines on K in Fig. 1(b).

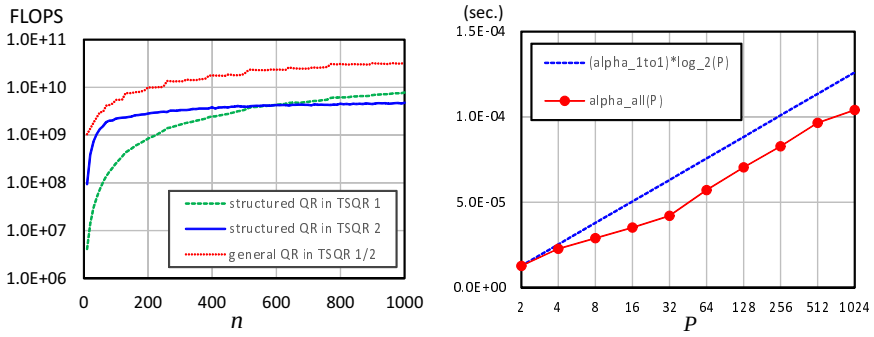
4.2 Results and discussion

We show the execution time and its breakdown of each program in Figs. 2(a) to (d), where n is varied and P and m are fixed. These graphs support our analysis that T_{diff} mostly comes from the difference between the communication time in Householder QR and the floating-point operation time of the structured QR factorizations in TSQR. Besides, the cases where TSQR is slower than Householder QR actually exist.

We next show the dependency of T_{diff} on m in Figs. 3(a) and (b), where $P = 1024$ and n was varied from 10 to 1000 by 10. As analyzed in the previous section, little dependency of T_{diff} on m can be seen from these graphs. We also show the relationship between T_{diff} and P in Figs. 4(a) and (b), where $m = 2,048,000$ and n was varied. It can be observed from them that T_{diff} is proportional to $\log_2 P$. These four graphs also clearly illustrate that TSQR becomes slower than Householder QR as n increases.

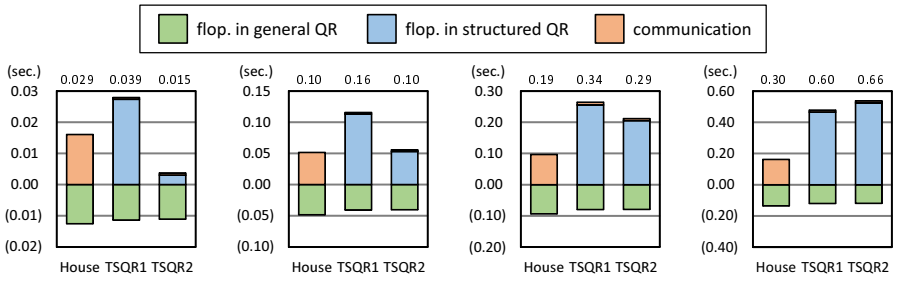
We then examine the estimation of T_{diff}. The results of the estimations are shown in Figs. 5(a) and (b), where est. 1 is the result by Eq. (3), est. 2 is that by Eq. (4), and est. 3 is that by Eq. (3) with γ_{st} replaced by γ_{ge}. Note that we used $\gamma_{\text{st}} = 1.43 \times 10^{-10}$ for TSQR 1, 2.22×10^{-10} for TSQR 2, and $\gamma_{\text{ge}} = 3.30 \times 10^{-11}$ in the estimations. These graphs shows that the estimations by Eq. (4) are more accurate than those by Eq. (3) with γ_{st}. The reason that the difference of the accuracy between these two kinds of estimation for TSQR 2 is smaller than that for TSQR 1 would be γ_{st} for TSQR 1 changes more wildly with n than that for TSQR 2 does. These results also indicate that the estimations using γ_{ge} often mislead that TSQR is almost always faster than Householder QR until n is unpractically large.

We finally list the results of the algorithm selections based on the estimation of T_{diff} in Table 2(a) and (b), where each figure means the number of correct selections. These tables show that estimation based on Eq. (4) gives reliable determinations except for the cases in which m/P is very large; additional investigation is required to make the reason why accuracy is bad in such cases. For TSQR 2, estimations based on Eq. (3) with γ_{st} are also accurate; this would be due to the behavior of γ_{st} in TSQR 2.



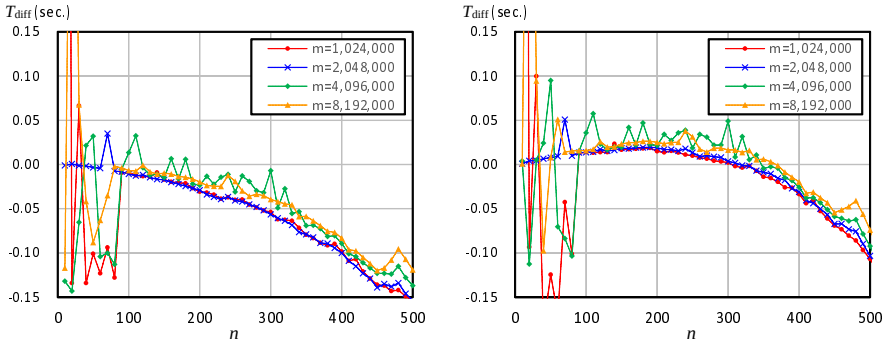
(a) Performance of each QR program. (b) Setup cost of the MPI routines on K.

Fig. 1. Basic performance data in numerical experiments.



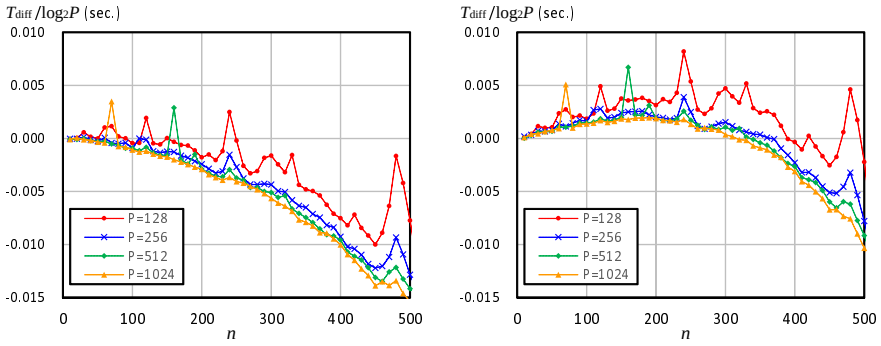
(a) $n = 100$ (b) $n = 300$ (c) $n = 500$ (d) $n = 700$

Fig. 2. Execution times and their breakdowns: n is varied ($P = 1024$, $m = 2,048,000$). The value on each bar is the total execution time.



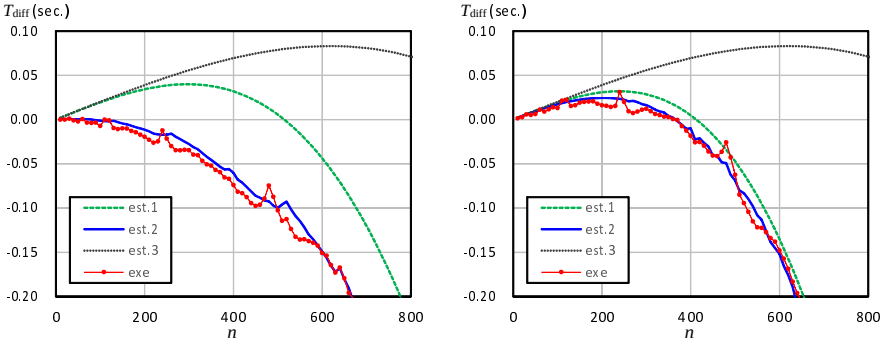
(a) $T_{\text{House}} - T_{\text{TSQR 1}}$ (b) $T_{\text{House}} - T_{\text{TSQR 2}}$

Fig. 3. Dependency of T_{diff} on m: $P = 1024$.



(a) $T_{\text{House}} - T_{\text{TSQR 1}}$ (b) $T_{\text{House}} - T_{\text{TSQR 2}}$

Fig. 4. Dependency of T_{diff} on P: $m = 2,048,000$.



(a) $T_{\text{House}} - T_{\text{TSQR 1}}$ (b) $T_{\text{House}} - T_{\text{TSQR 2}}$

Fig. 5. Estimations of T_{diff}: $P = 1024$ and $m = 2,048,000$. est. 1 is by Eq. (3), est. 2 is by Eq. (4), and est. 3 is by Eq. (3) with γ_{ge} instead of γ_{st}.

Table 2. The number of correct prediction of the faster algorithm (House v.s. TSQR 1 or 2): the left figure in each cell was by Eq. (3) with γ_{st} and the right one was by Eq. (4). In each setting of P and m, 100 cases of $n(= 10, 20, \ldots, 1000)$ were tested.

(a) House v.s. TSQR 1

$m \backslash P$	128	256	512	1024
1,024,000	54 94	51 92	56 93	51 94
2,048,000	59 93	51 94	50 91	51 94
4,096,000	82 72	62 93	55 92	55 90
8,192,000	76 40	80 75	58 91	51 94

(b) House v.s. TSQR 2

$m \backslash P$	128	256	512	1024
1,024,000	93 98	97 96	86 90	83 87
2,048,000	93 94	95 100	92 96	90 94
4,096,000	92 87	95 96	89 93	89 93
8,192,000	82 77	92 86	91 93	94 98

5 Related work

The TSQR algorithm has been evaluated in several environments [3, 4], however what is described in this parer, particularly the dependency of the difference of the execution time on n and the ways of predicting it, has rarely been reported as far as the authors know. The CAQR algorithm [1] and the tile QR [5] algorithm are other well-known communication algorithms, however they have mainly been evaluated for the QR factorization of (nearly) square matrices in existing works.

6 Conclusion

We analyzed the parallel execution times of Householder QR and TSQR by using the realistic model with distinguished floating-point throughputs. We also pointed out that there are considerable cases where TSQR is slower than Householder QR, and presented a way to estimate the difference of the execution times. Almost all of the results of the numerical experiments on the K computer supported our analyses and showed reliability of our estimation approach. This study will lead to an automatic algorithm selection between Householder QR and TSQR.

But the following future works are of more practical importance. Experiments in other environments, in particular, multi-cpu clusters where the cost of intra-node communications differs from that of the inter-node ones, should be considered. In addition, the cases where explicit Q is required and where other matrices need to be updated by the same transformations also should be studied. A more important work is extending our analyses to the case in which the panel blocking technique is employed, so-called CAQR [1], which is expected to give better performance when n becomes large.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments. 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. Demmel, J., Grigori, L., Hoemmen, M. and Langou, J.: Communication-optimal parallel and sequential QR and LU factorizations. SIAM J. SCI. Comput. **34** (2012) 206–239
2. Goulb, G. and Van Loan, C.: Matrix Computations. 3rd ed. Johns Hopkins Univ. Press (1996)
3. Agullo, E., Coti, C., Dongarra, J., Herault, T. and Langou, J.: QR factorization of tall and skinny matrices in a grid computing environment. 24th IEEE International Parallel and Distributed Processing Symposium (2010) 1–11
4. Constantine, G. and Gleich, D.: Tall and skinny QR factorizations in MapReduce architectures. Proceedings of the 2nd international workshop on MapReduce and its applications (2011) 43–50
5. Song, F., Ltaief, H., Hadri, B. and Dongarra, J.: Scalable tile communication-avoiding QR factorization on multicore cluster systems. Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (2010) 1–11