A communication-avoiding thick-restart Lanczos method on a distributed-memory system

Ichitaro Yamazaki* and Kesheng Wu

Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Abstract. The Thick-Restart Lanczos (TRLan) method is an effective method for solving large-scale Hermitian eigenvalue problems. On a modern computer, communication can dominate the solution time of TRLan. To enhance the performance of TRLan, we develop CA-TRLan that integrates communication-avoiding techniques into TRLan. To study the numerical stability and solution time of CA-TRLan, we conduct numerical experiments using both synthetic diagonal matrices and matrices from the University of Florida sparse matrix collection. Our experimental results on up to $1,024$ processors of a distributed-memory system demonstrate that CA-TRLan can achieve speedups of up to three over TRLan while maintaining numerical stability.

Keywords: thick-restart Lanczos, communication-avoiding.

1 Introduction

TRLan implements a thick-restart Lanczos method [12] on a distributed-memory system. It is effective for computing a few exterior eigenvalues λ and their corresponding eigenvectors v of a Hermitian matrix A:

$$Av = \lambda v.$$

On a modern computer, the time spent on communication (i.e., data access through memory hierarchy and data transfer between processors) can dominate the solution time of TRLan. To improve the performance of TRLan, in this paper, we study techniques to avoid communication. Several techniques to avoid communication of Krylov subspace methods have been proposed [5], but their performance on a distributed-memory system, especially for solving eigenvalue problems, has not yet been studied.

The rest of the paper is organized as follows: We first review TRLan in Section 2. Then, in Section 3, we develop a communication-avoiding version of TRLan (CA-TRLan). The experimental results of CA-TRLan are presented in Section 4. Finally, we conclude with final remarks in Section 5.

* Corresponding author: ic.yamazaki@gmail.com

2 Thick-Restart Lanczos method

Given a starting vector q, the i-th iteration of the Lanczos method [6] computes a new orthonormal basis vector q_{i+1} of a Krylov subspace,

$$\mathcal{K}_{i+1}(q, A) \equiv \text{span}\{q, Aq, A^2q, \ldots, A^iq\} \equiv \text{span}\{q_1, q_2, \ldots, q_{i+1}\}.$$

These basis vectors satisfy the relation

$$AQ_i = Q_iT_i + \beta_i q_{i+1} e_i^H, \tag{1}$$

where $Q_i = [q_1, q_2, \ldots, q_i]$, $\beta_i = q_{i+1}^H Aq_i$, e_i is the i-th column of the i-dimensional identity matrix I_i, $T_i = Q_i^H AQ_i$ is an $i \times i$ Rayleigh-Ritz projection of A onto $\mathcal{K}_i(A, q)$, and the superscript ^{H} indicates the conjugate transpose. Then, an approximate eigenpair $(\theta, x = Q_iy)$ of A is computed from an eigenpair (θ, y) of T_i, where θ and x are referred to as a Ritz value and Ritz vector, respectively. It is well known that Ritz values converge to exterior eigenvalues of A with a subspace dimension i that is much smaller than the dimension n of A [7, 8].

A key feature that distinguishes the Lanczos method from other subspace projection methods is that T_i of (1) is symmetric tridiagonal:

$$T_i = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \alpha_{i-1} & \beta_{i-1} \\ & & & \beta_{i-1} & \alpha_i \end{pmatrix}.$$

This leads to the following simple three-term recurrence:

$$\beta_i q_{i+1} = Aq_i - \alpha_i q_i - \beta_{i-1} q_{i-1}. \tag{2}$$

Subsequently, in theory, the new basis vector q_{i+1} can be computed by orthonormalizing the vector Aq_i against two preceding basis vectors, q_{i-1} and q_i. However, in finite precision arithmetic, the orthogonality among the basis vectors is lost even after a small number of iterations. To maintain orthogonality, TRLan reorthogonalizes the new basis vector q_{i+1} against all the previous vectors Q_i using a Classical Gram-Schmidt (CGS) procedure. As the basis size $i+1$ grows, this procedure becomes expensive in terms of both computation and storage.

To reduce the costs of computing a large subspace, TRLan restarts the iteration after a fixed number $m+1$ of the basis vectors is computed. At the $(j-1)$-th restart, TRLan sets k_j computed Ritz vectors and the last basis vector q_{m+1} as the first $k_j + 1$ basis vectors of the next projection subspace. To compute the (k_j+2)-th basis vector, TRLan computes $A\widehat{q}_{k_j+1}$ and orthonormalizes it against the previous basis vectors,[1]

$$\widehat{\beta}_{k_j+1}\widehat{q}_{k_j+2} = A\widehat{q}_{k_j+1} - \widehat{Q}_{k_j}(\widehat{Q}_{k_j}^H A\widehat{q}_{k_j+1}) - \widehat{q}_{k_j+1}(\widehat{q}_{k_j+1}^H A\widehat{q}_{k_j+1}), \tag{3}$$

[1] A hat distinguishes the i-th basis vector $\widehat{q}_i$ computed after the restart from the i-th basis vector q_i computed before the restart.

where $\widehat{Q}_{k_j}^H A\widehat{q}_{k_j+1} = \beta_m h$, $h = Y_{k_j}^H e_m$, and Y_{k_j} are the eigenvectors of T_m corresponding to the kept Ritz values.

In general, at the i-th iteration of the j-th restart-loop, the basis vectors satisfy the relation:

$$A\widehat{Q}_{k_j+i} = \widehat{Q}_{k_j+i}\widehat{T}_{k_j+i} + \widehat{\beta}_{k_j+i}\widehat{q}_{k_j+i+1}e_{k_j+i}^H,$$

where $\widehat{T}_{k_j+i} = \widehat{Q}_{k_j+i}^H A\widehat{Q}_{k_j+i}$ is of the form

$$\widehat{T}_{k_j+i} = \begin{pmatrix} D_{k_j} & \beta_m h & & & & \\ \beta_m h^H & \widehat{\alpha}_{k_j+1} & \widehat{\beta}_{k_j+1} & & & \\ & \widehat{\beta}_{k_j+1} & \widehat{\alpha}_{k_j+2} & \widehat{\beta}_{k_j+2} & & \\ & & \widehat{\beta}_{k_j+2} & \ddots & \ddots & \\ & & & \ddots & \widehat{\alpha}_{k_j+i-1} & \widehat{\beta}_{k_j+i-1} \\ & & & & \widehat{\beta}_{k_j+i-1} & \widehat{\alpha}_{k_j+i} \end{pmatrix}, \tag{4}$$

and D_{k_j} is a diagonal matrix whose diagonal elements are the kept Ritz values. In theory, the three-term recurrence is invalid only for computing q_{k_j+2} and is resumed afterward. However, to maintain orthogonality in finite precision arithmetic, TRLan reorthogonalizes each basis vector q_{k_j+i+1} against all the previous vectors. This full-reorthogonalization procedure and the matrix-vector multiplication often dominate the iteration time of TRLan. A detailed description of TRLan can be found in [11]. Effective auto-tuning schemes have been proposed to dynamically select the next subspace dimension m and the Ritz vectors to keep at each restart [12].

3 Communication-avoiding TRLan

To compute q_{k_j+i+1} at the i-th iteration after the $(j-1)$-th restart, TRLan requires the *intra-processor* communication of the local portion of A and Q_{k_j+i} through the memory hierarchy, and also requires the *inter-processor* communication of all the required non-local data between the processors. This communication can dominate the iteration time of TRLan. To enhance the performance of TRLan, we develop CA-TRLan, which integrates techniques to avoid communication of Krylov subspace methods [5] into TRLan. The main idea is to compute a set of s basis vectors at each iteration. Below, we describe the t-th iteration of CA-TRLan to generate the next s basis vectors for the j-th restart-loop, where the basis vectors Q_{m_t+1} have been computed (i.e., $m_t = k_j + 1 + (t-1)s$).[2]

3.1 Krylov subspace generation

Given a starting vector q_{m_t+1}, the t-th iteration of CA-TRLan first generates basis vectors of the Krylov subspace $\mathcal{K}_{s+1}(q_{m_t+1}, A)$.[3] One option is to generate

[2] The last iteration generates $m - m_{t-1}$ basis vectors.

[3] The first starting vector q_{m_1+1} is q_{k_j+2} computed by the full-orthogonalization (3).

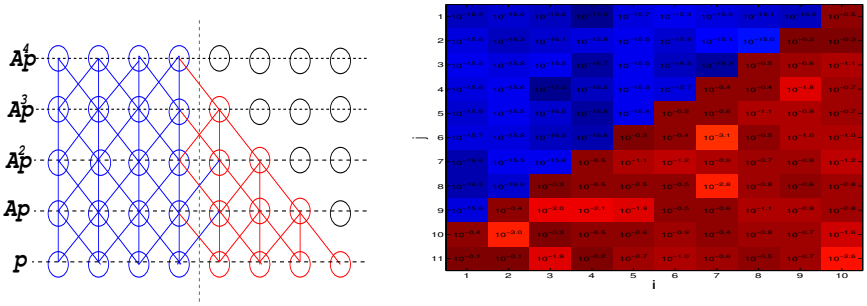

Fig. 1. Matrix-power kernel on a tridiagonal A (left) and orthogonality levels (right).

the monomial basis vectors $q_{m_t+1}, Aq_{m_t+1}, A^2q_{m_t+1}, \ldots, A^sq_{m_t+1}$, but the vector $A^sq_{m_t+1}$ converges to the principal eigenvector of A, leading to numerical instability in finite precision arithmetic. To avoid this instability, as suggested in [1, 5], CA-TRLan generates the Newton basis vectors $P_+^{(t)} = [p_1^{(t)}, p_2^{(t)}, \ldots, p_{s+1}^{(t)}]$, where $p_i^{(t)} = \prod_{\ell=1}^{i-1}(A - \theta_\ell I)q_{m_t+1}$, and θ_ℓ are the Ritz values computed at the $(j-1)$-th restart in the Leja ordering. Notice that

$$AP^{(t)} = P_+^{(t)} B_+^{(t)}, \tag{5}$$

where $P^{(t)} = [p_1^{(t)}, p_2^{(t)}, \ldots, p_s^{(t)}]$ and $B^{(t)}$ is an $(s+1) \times s$ matrix whose diagonal entries are $\theta_1, \theta_2, \ldots, \theta_s$ and subdiagonal entries are ones. In order to obtain the shifts θ_ℓ, TRLan (without avoiding communication) is used for the first restart-loop of CA-TRLan.

The left figure in Fig. 1 illustrates our implementation of the matrix-power kernel to generate a set of s basis vectors $P_+^{(t)}$ for a tridiagonal matrix A (i.e., $s = 4$). In the figure, blue elements are the local portion of $P_+^{(t)}$, which must be computed by a processor, and edges between elements represent the data dependency. Our implementation first uses a k-way edge partitioning algorithm of METIS[4] to partition and distribute A into a block row format. The basis vectors $P^{(t)}$ are distributed in the conforming format. Now, at each t-th iteration to compute $P_+^{(t)}$, each processor first gathers all the required elements of q_{m_t+1} from the neighboring processors (the red elements of p in Fig. 1), and then it computes the local portion of $P_+^{(t)}$ without further communication. This requires the computation and storage of additional elements (the red elements in Fig. 1), but needs only one synchronization to compute the s vectors. The rows of A corresponding to the red elements of p in Fig. 1 are duplicated by the neighboring processors. A detailed discussion of the matrix-power kernel can be found in [4].

[4] http://glaros.dtc.umn.edu/gkhome/metis/metis.

3.2 Orthogonalization

Once the Newton basis vectors $P_+^{(t)}$ are computed as described in Section 3.1, CA-TRLan orthogonalizes $P_+^{(t)}$ and generates the orthonormal basis vectors $Q_+^{(t)} = [q_1^{(t)}, q_2^{(t)}, \ldots, q_{s+1}^{(t)}] = [q_{m_t+1}, q_{m_t+2}, \ldots, q_{m_t+s+1}]$. This is done by first orthogonalizing $P_+^{(t)}$ against the previous basis vectors Q_{m_t} and then orthogonalizing among the vectors in $P_+^{(t)}$. Note that the first vector $p_1^{(t)}$ is the last basis vector $q_{s+1}^{(t-1)}$ from the previous iteration, and it is already orthogonal to Q_{m_t}. Furthermore, because of the three-term recurrence (2), we have

$$p_{i+1}^{(t)} \in \text{span}\{q_{m_t-i+1}, q_{m_t-i+2}, \ldots, q_{m_t+i+1}\} \tag{6}$$

for $i = 1, 2, \ldots, s$. Hence, in theory, we can obtain the next basis vector q_{m_t+i+1} by orthonormalizing $p_{i+1}^{(t)}$ against the previous $2 \cdot i$ basis vectors. To illustrate this, the right figure of Fig. 1 shows the orthogonality level $r_{j,i+1}^{(t)}$ of $p_{i+1}^{(t)}$ against $q_j^{(t-1)}$ (i.e., $r_{j,i+1}^{(t)} = |q_j^{(t-1)H} p_{i+1}^{(t)}| / \|p_{i+1}^{(t)}\|_2$) at the second iteration of the second restart-loop for computing the 100 smallest eigenvalues of $\text{diag}(1^2, 2^2, \ldots, 10000^2)$ with $s = 10$.[5] We clearly see that $r_{j,i+1}^{(t)}$ diminishes for $j \leq s - i$. Only exception is at the first iteration (i.e., $t = 1$), where the full-orthogonalization is needed because $p_1^{(1)}$ is computed by (3), and $p_{i+1}^{(1)}$ does not satisfy (6).

Based on the above observations, the first step of our orthogonalization procedure uses CGS to orthogonalizes $P_+^{(t)}$ against the previous vectors Q_{m_t}. For this, xGEMM of level-3 BLAS is used once to compute $\bar{R}_-^{(t)} := Q_+^{(t-1)H} P_-^{(t)}$ and one more time to compute $\bar{P}_-^{(t)} := P_-^{(t)} - Q_+^{(t-1)} \bar{R}_-^{(t)}$.[6] The vector $p_{i+1}^{(t)}$ is orthogonalized against all the vectors in $Q_+^{(t-1)}$ even though it must be orthogonalized against only $i + 1$ vectors. This improves the data locality while increasing the operation count. TRLan also uses CGS for reorthogonalization, but it orthogonalizes one vector at a time using xGEMV of level-2 BLAS.

The second step of the orthogonalization procedure computes the QR factorization of $\bar{P}_-^{(t)}$ (i.e., $Q_-^{(t)} R_-^{(t)} = \bar{P}_-^{(t)}$). For this, we have implemented the following four approaches, where $\bar{P}_-^{(t)}$ and $Q_-^{(t)}$ are distributed among all the available p processors in a row block layout, while $R_-^{(t)}$ is duplicated on each processor:

- **CGS-approach**: Similarly to TRLan, each vector of $\bar{P}_-^{(t)}$ is orthogonalized against the previous vectors based on CGS using xGEMV of level-2 BLAS. Communication is needed for orthogonalizing and normalizing each vector.
- **CholeskyQR-approach**: xGEMM is first used to compute $\bar{P}_-^{(t)H} \bar{P}_-^{(t)}$ and then xPORTRF of LAPACK is used to compute its Cholesky factor $R_-^{(t)}$; i.e., $R_-^{(t)H} R_-^{(t)} = \bar{P}_-^{(t)H} \bar{P}_-^{(t)}$. Finally, xTRSM of LAPACK is used to compute $Q_-^{(t)}$

[5] A detailed description of the numerical experiment can be found in Section 4.

[6] For the full-orthogonalization at $t = 1$, we compute $\bar{R}_-^{(t)} := Q_{m_t}^H P_-^{(t)}$.

by solving the triangular system $R_-^{(t)H} Q_-^{(t)H} = \bar{P}_-^{(t)H}$. This approach requires about the same number of arithmetic operations as the CGS-approach. However, when the window size is much smaller than the local matrix size, i.e., $s \ll n/p$, most of the computation is performed using level-3 BLAS. Moreover, this approach achieves the lower-bound of the communication volume and message count for the QR factorization. Even though this approach was successful in our numerical experiments, it may be numerically unstable when A is ill-conditioned. More discussion of the numerical stability, and the computation and communication bounds can be found in [3, 9].

- **ScaLAPACK-approach**: The ScaLAPACK subroutines PxGEQRF and PxORGQR, which implement a numerically stable Householder QR factorization, were used. Even though ScaLAPACK expects $\bar{P}_-^{(t)}$ in a 2D block cyclic format, we used the row block layout to call ScaLAPACK in our numerical experiments. We also converted the matrix into a 1D block cyclic format, but the overhead of conversion can be significant, while the expected performance gain is small.
- **TSQR-approach**: A communication-avoiding QR algorithm for tall-skinny matrices [3] is used. This algorithm obtains the lower-bound of communication and is as numerically stable as ScaLAPACK.

The resulting basis vectors satisfy the following relation:

$$P_{m_{t+1}+1} = Q_{m_{t+1}+1}\, R_{m_{t+1}+1}, \tag{7}$$

where $P_{m_{t+1}+1} = [Q_{k_j+1}, P_-^{(1)}, P_-^{(2)}, \ldots, P_-^{(t)}]$, $R_{m_{t+1}+1} = \begin{pmatrix} R_{m_t+1} & \widehat{R}_-^{(t)} \\ & R_-^{(t)} \end{pmatrix}$, $R_{m_1+1} = I_{k_j+2}$, and $\widehat{R}_-^{(t)} = Q_{m_t+1}^H P_-^{(t)}$. In theory, we have $\widehat{R}_-^{(t)} = \begin{pmatrix} 0_{m_t \times s} \\ \bar{R}_-^{(t)} \end{pmatrix}$ for $t \geq 2$, where $0_{m_t \times s}$ is an $m_t \times s$ zero matrix, and $\bar{R}_-^{(t)}$ has the nonzero structure indicated by Fig. 1. To maintain orthogonality, however, CA-TRLan performs the full-reorthogonalization of $Q_-^{(t)}$ using the same two-step procedure; i.e., $Q_-^{(t)}$ is first reorthogonalized against Q_{m_t+1} and then reorthogonalized among the vectors in $Q_-^{(t)}$. As we will show in Section 3.3, only the bidiagonal elements of $R_-^{(t)}$ are needed in the remaining steps of CA-TRLan. Hence, after the reorthogonalization, only these required elements of $R_-^{(t)}$ are updated.

3.3 Tridiagonal matrix computation

Finally, to expand the tridiagonal matrix $\widehat{T}_{m_{t+1}}$ of (4), we compute the projected matrix $T_+^{(t)} = Q_+^{(t)H} A Q^{(t)}$, where $Q^{(t)} = [q_1^{(t)}, q_2^{(t)}, \ldots, q_s^{(t)}]$. By (5) and (7), we have $T_+^{(t)} = R_+^{(t)} B^{(t)} (R^{(t)})^{-1}$, where $R_+^{(t)}$ and $R^{(t)}$ are the $(s+1) \times (s+1)$ and $s \times s$ lower-right submatrices of $R_{m_{t+1}+1}$ and $R_{m_{t+1}}$ of (7), respectively [5, Section 4.2.2]. Hence, the elements of $T_+^{(t)}$ are computed by the following simple

recursion: for $i = 1, 2, \ldots, s$,

$$\begin{cases} \alpha_i^{(t)} = (\theta_i r_{i,i}^{(t)} + r_{i,i+1}^{(t)} - \beta_{i-1}^{(t)} r_{i-1,i}^{(t)})/r_{i,i}^{(t)}, \\ \beta_i^{(t)} = r_{i+1,i+1}^{(t)}/r_{i,i}^{(t)}, \end{cases}$$

where $\alpha_i^{(t)} := \alpha_{m_t+i}$, $\beta_i^{(t)} := \beta_{m_t+i}$, $\beta_0^{(t)} = 0$, and $r_{i,j}^{(t)}$ is the (i,j)-th element of $R_+^{(t)}$. Notice that only the bidiagonal elements of $R_+^{(t)}$ are needed.

4 Numerical experiments

We first study the effects of the window size s on the performance of CA-TRLan. Table 1 shows the results of computing the 100 smallest eigenvalues of the diagonal matrix $A_k(n) = \mathrm{diag}(1^k, 2^k, \ldots, n^k)$ of dimension $n = 10,000$ with $k = 1, 2$, or 3 on a single core of a Cray-XT4 machine at NERSC. The dimension of the projection subspace is fixed at $m = 200$ for $k = 1$ and 2, and at $m = 400$ for $k = 3$. A Ritz pair (θ, x) is considered to be converged when $\|Ax - \theta x\|_2 \leq 10^{-16}\|A\|_2$ and $\|A\|_2$ is approximated by the largest modulus of the computed Ritz values. In the table, "rest." is the number of restarts; "ops" is the number of matrix-operations; "res. norm" is the relative residual norm of the 100-th smallest computed eigenvalue; "time" shows the times required for the matrix-operations, orthogonalization, and restarts, and the total solution time in seconds. Under "time," we also show the time spent on the QR factorization of $\bar{P}_-^{(t)}$ in parentheses.

In the table, we see that the solution accuracy of CA-TRLan with s up to 20 was similar to that of TRLan in most cases. The only exceptions were when the CGS- or CholeskyQR-approach was used with $s = 15$ on A_3. In these cases, CA-TRLan failed to converge due to numerical instability. On the other hand, the ScaLAPACK- and TSQR-approaches were numerically stable with s up to 20. We also see that the orthogonalization time of CA-TRLan was much less than that of TRLan due to the reduced intra-processor communication on the single core. Furthermore, the first step of the orthogonalization procedure dominated the orthogonalization time. A larger value of s improved the data locality and reduced the time spent in this first step, but it increased the time required for the second step, which must orthogonalize the vectors in a larger window. Even though the TSQR-approach was faster than the ScaLAPACK-approach, these two approaches were significantly slower than the CGS- and CholeskyQR-approaches, especially with a larger value of s.

We next examine the parallel scalability of CA-TRLan. The left plot of Fig. 2 shows the solution times of CA-TRLan and TRLan computing the smallest 100 eigenvalues of $A_3(10,000)$. We used the CholeskyQR-based orthogonalization and the fixed parameters $m = 400$ and $s = 10$. The number above each bar is the speedup gained by CA-TRLan over TRLan using the same number of processors. A significant speedup was obtained by CA-TRLan on one processor due to the optimized intra-processor communication of level-3 BLAS. However, as the processor count increased (e.g., 4 to 64 processors), the local matrix size decreased,

s	rest.	ops	res. norm	time ops	orth	rest.	total
— results with $A_1(10,000)$ —							
TRLan							
	34	2.4K	9.3×10^{-12}	0	10 (0)	2	13
CA-TRLan with CGS-approach							
5	34	2.5K	$9.9 \cdot 10^{-12}$	0	6 (0)	2	9
10	34	2.5K	$9.7 \cdot 10^{-12}$	0	5 (1)	2	8
15	34	2.6K	$1.4 \cdot 10^{-11}$	0	5 (1)	2	8
20	34	2.7K	$1.7 \cdot 10^{-11}$	0	5 (1)	2	8
CA-TRLan with CholeskyQR-approach							
5	34	2.5K	$9.0 \cdot 10^{-12}$	0	6 (0)	3	9
10	34	2.5K	$1.0 \cdot 10^{-11}$	0	5 (0)	3	8
15	34	2.6K	$1.4 \cdot 10^{-11}$	0	5 (1)	3	8
20	34	2.7K	$2.5 \cdot 10^{-11}$	0	5 (1)	3	8
CA-TRLan with ScaLAPACK-approach							
5	34	2.5K	$1.6 \cdot 10^{-11}$	0	7 (1)	2	10
10	34	2.5K	$1.8 \cdot 10^{-11}$	0	6 (1)	2	9
15	34	2.6K	$4.2 \cdot 10^{-11}$	0	7 (2)	2	9
20	34	2.7K	$4.0 \cdot 10^{-11}$	0	7 (3)	2	10

s	rest.	ops	res. norm	time ops	orth	rest.	total
— results with $A_2(10,000)$ —							
TRLan							
	377	21.0K	1.0×10^{-8}	1	94 (0)	29	125
CA-TRLan with CGS-approach							
5	374	21.7K	$8.9 \cdot 10^{-8}$	1	59 (3)	29	92
10	361	22.3K	$9.0 \cdot 10^{-8}$	1	46 (5)	28	78
15	376	23.7K	$9.6 \cdot 10^{-8}$	1	50 (9)	29	83
20	375	25.8K	$1.0 \cdot 10^{-7}$	1	53 (12)	29	86
CA-TRLan with CholeskyQR-approach							
5	370	21.7K	$9.0 \cdot 10^{-8}$	1	59 (3)	29	91
10	362	22.5K	$1.0 \cdot 10^{-7}$	1	45 (4)	28	77
15	367	23.7K	$9.1 \cdot 10^{-8}$	1	46 (5)	29	78
20	364	25.2K	$9.6 \cdot 10^{-8}$	1	46 (6)	29	78
CA-TRLan with ScaLAPACK-approach							
5	370	21.8K	$1.8 \cdot 10^{-7}$	1	65 (9)	29	97
10	360	22.2K	$2.1 \cdot 10^{-7}$	1	55 (14)	28	87
15	359	23.3K	$2.2 \cdot 10^{-7}$	1	61 (21)	28	93
20	374	25.6K	$2.1 \cdot 10^{-7}$	1	70 (30)	29	103

s	rest.	ops	res. norm	time ops	orth	rest.	total
— results with $A_3(10,000)$ —							
TRLan							
	2.4K	192K	$8.7 \cdot 10^{-4}$	6	1722 (0)	254	2002
CA-TRLan with CGS-approach							
5	2.6K	210K	$8.4 \cdot 10^{-4}$	12	1115 (27)	265	1445
10	2.3K	200K	$8.6 \cdot 10^{-4}$	10	763 (49)	249	1070
15	−−	−−	−−	−−	−−	−−	−−
CA-TRLan with CholeskyQR-approach							
5	2.6K	210K	$8.5 \cdot 10^{-4}$	12	1109 (27)	265	1439
10	2.5K	212K	$8.8 \cdot 10^{-4}$	11	799 (36)	262	1145
15	−−	−−	−−	−−	−−	−−	−−

s	rest.	ops	res. norm	time ops	orth	rest.	total
CA-TRLan with ScaLAPACK-approach							
5	2.7K	213K	$1.7 \cdot 10^{-3}$	12	1208 (85)	266	1541
10	2.2K	199K	$2.4 \cdot 10^{-3}$	10	836 (129)	249	1141
15	2.4K	206K	$5.4 \cdot 10^{-3}$	11	869 (189)	253	1182
20	2.6K	235K	$2.5 \cdot 10^{-2}$	12	984 (277)	262	1313
CA-TRLan with TSQR-approach							
5	2.7K	212K	$8.7 \cdot 10^{-4}$	12	1163 (67)	266	1497
10	2.8K	227K	$9.2 \cdot 10^{-4}$	12	923 (98)	271	1262
15	2.6K	213K	$1.6 \cdot 10^{-3}$	11	813 (108)	258	1134
20	2.6K	234K	$5.6 \cdot 10^{-3}$	12	841 (140)	258	1169

Table 1. Numerical results of CA-TRLan on synthetic diagonal matrices.

and so did the improvement gained using level-3 BLAS. Since the solution time was dominated by computation and intra-processor communication on a small number of processors, the speedups gained by CA-TRLan decreased. However, as the processor count increased further, inter-processor communication started to dominate the solution time (labeled by "MPI" in Fig. 2). CA-TRLan avoided some of the inter-processor communication, and the improvement gained by CA-TRLan started to increase. The right plot of Fig. 2 shows the timing results of CA-TRLan and TRLan computing the smallest 30 eigenvalues of $A_2(10,000)$ with $m = 80$ and $s = 15$. In comparison to the left plot, the right plot shows more significant performance advantage of CA-TRLan because communication started to dominate the solution time of TRLan much sooner.

Finally, Table 2 shows the results of CA-TRLan computing the 30 smallest eigenvalues of four matrices (#1 Andrews, #2 torsion1, #3 cfd1, and #4 finan512) from the University of Florida sparse Matrix Collection (UFMC). For each ℓ-th matrix, we show $\#\ell(n, nnz, \text{cond})$, where "$n$" is the matrix dimension in thousands, "nnz" is the number of nonzeros in thousands, and "cond" is the 1-norm condition number estimate computed using the MATLAB subroutine

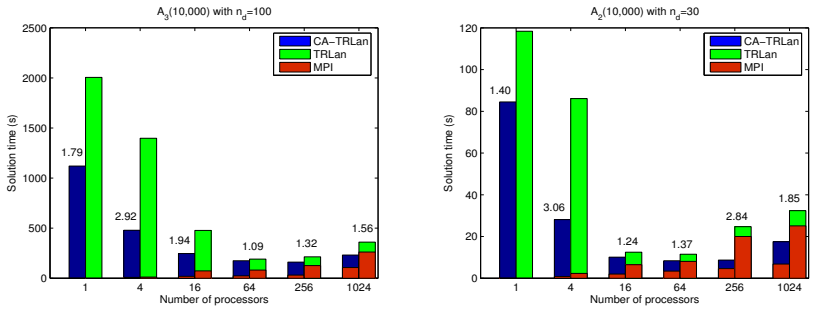

Fig. 2. Performance of CA-TRLan computing n_d eigenpairs of diagonal matrices.

s	surf/vol A^s	surf/vol A	rest.	ops	res. norm	time op	time ort	time tot
#1$(60, 760, 2.5 \times 10^{17})$, $ln(1.8, 2.0)$ $lnz(2.3, 2.5)$								
TRLan								
	0.0	0.0	24	865	$3.3 \cdot 10^{-14}$	2	1	4
CA-TRLan								
1	0.0	0.0	24	865	$3.1 \cdot 10^{-14}$	2	2	4
5	17.1	25.2	24	917	$3.1 \cdot 10^{-14}$	2	1	3
10	−−	32.0	24	972	$3.3 \cdot 10^{-14}$	5	1	6
#2$(40, 198, 31)$, $ln = (1.2, 1.3)$ $lnz(5.6, 6.3)$								
TRLan								
	0.0	0.0	118	3046	$1.1 \cdot 10^{-14}$	4	3	9
CA-TRLan								
1	0.0	0.0	116	3018	$1.1 \cdot 10^{-14}$	4	5	9
5	0.5	0.5	111	3234	$1.0 \cdot 10^{-14}$	1	1	3
10	1.2	1.3	116	3686	$1.2 \cdot 10^{-14}$	1	1	3
#3$(71, 1.8, 1.3 \times 10^{6})$, $ln(2.1, 2.3)$ $lnz(53, 59)$								
TRLan								
	0.0	0.0	509	10318	$7.9 \cdot 10^{-15}$	17	15	36
CA-TRLan								
1	0.0	0.0	644	12245	$9.4 \cdot 10^{-15}$	22	24	49
5	4.3	3.9	510	11198	$8.3 \cdot 10^{-15}$	10	10	23
10	11.6	12.2	587	14415	$1.5 \cdot 10^{-15}$	19	17	38
#4$(75, 597, 98)$, $ln = (2.2, 2.2)$ $lnz(18, 19)$								
TRLan								
	0.0	0.0	78	1921	$1.2 \cdot 10^{-14}$	3	3	7
CA-TRLan								
1	0.0	0.0	77	1910	$1.1 \cdot 10^{-14}$	3	4	8
5	0.5	0.4	79	2112	$1.1 \cdot 10^{-14}$	1	1	3
10	2.6	2.3	78	2306	$3.8 \cdot 10^{-14}$	1	1	3

Table 2. Numerical results of CA-TRLan with matrices from UFMC collection.

condest. Furthermore, "ln" and "lnz" show the minimum and maximum local dimensions and numbers of nonzeros in thousands. The experiments were conducted on 32 processors with the CholeskyQR-based orthogonalization and $(m, s) = (80, 10)$. In the table, "surf/vol" is the overhead of the matrix-power kernel discussed in Section 3.1. Specifically, under "A," we show the maximum ratio of the number of duplicated nonzeros of A over the number of local nonzeros of A (corresponding to the red and blue elements, respectively, in Fig. 1) on one processor. Under "A^s," we show the same ratio obtained computing the k-way edge partition of explicitly computed A^s.[7] We see that the these two ratios are similar for the same value of s indicating that the partition of A was as good as that obtained on A^s for these matrices. The numerical and timing results are those with the partition of A. CA-TRLan obtained solution accuracy similar to that of TRLan while achieving speedups of up to 3.54.

[7] In the table "−−" indicates insufficient memory to compute A^s.

5 Conclusion

We have studied techniques to avoid communication of TRLan on a distributed-memory system. The experimental results demonstrated that these techniques can significantly reduce the solution time of TRLan while maintaining numerical stability. Our matrix-power kernel has not yet been optimized for intra-processor communication, and for this, we are planning to integrate the optimized sparse kernel interface (OSKI) [10] into CA-TRLan. Furthermore, partitioning algorithms to enhance matrix-power kernels (e.g., [2]) is an important computational kernel of CA-TRLan, and we would like to study the effectiveness of such an algorithm for computing eigenpairs of general matrices with CA-TRLan.

Acknowledgements

We gratefully thank Mark Hoemmen, Magnus Gustafsson, James Demmel, and Julien Langou for helpful discussions. This research was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. D-AC02-05CH11231, and the NSF PLASMA grant (#CCF-0811642) and the Microsoft PLASMA grant (MSRER RPD v3.1).

References

1. Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA Journal of Numerical Analysis*, 14:563–581, 1994.
2. E. Carson, N. Knight, and J. Demmel. Hypergraph partitioning for computing matrix powers. In *the proceedings of SIAM workshop on combinatorial scientific computing*, 2011.
3. J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. Technical Report UCB/EECS-2008-89, EECS Department, University of California, Berkeley, 2008.
4. J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in sparse matrix computations. In *the proceedings of IEEE international symposium on parallel and distributed processing (IPDPS)*, pages 1–12, 2008.
5. M. Hoemmen. *Communication-avoiding Krylov subspace methods.* PhD thesis, University of California, Berkeley, 2010.
6. C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–281, 1950.
7. B. Parlett. *The symmetric eigenvalue problem.* Classics in Applied Mathematics. SIAM, Philadelphia, PA, 1998.
8. Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Classics in Applied Mathematics. SIAM, Philadelphia, PA, 2 edition, 2003.
9. A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23:2165–2182, 2002.

10. R. Vuduc, J. Demmel, and K. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. In *Proc. SciDAC, J. Physics: Conf. Ser.*, volume 16, pages 521–530, 2005. Software available at http://bebop.cs.berkeley.edu/oski/.
11. K. Wu and H. Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Mat. Anal. Appl.*, 22:602–616, 2000.
12. I. Yamazaki, Z. Bai, H. Simon, L.-W. Wand, and K. Wu. Adaptive projection subspace dimension for the thick-restart Lanczos method. *ACM Trans. Math. Softw.*, 37, 2010. Software available at https://codeforge.lbl.gov/projects/trlan/.