2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)

Matrix Powers Kernels for Thick-Restart Lanczos with Explicit External Deflation

Zhaojun Bai†, Jack Dongarra*, Ding Lu†, and Ichitaro Yamazaki*
†The University of California, Davis, USA
*The University of Tennessee, Knoxville, USA

***Abstract*—Some scientific and engineering applications need to compute a large number of eigenpairs of a large Hermitian matrix. Though the Lanczos method is effective for computing a few eigenvalues, it can be expensive for computing a large number of eigenpairs (e.g., in terms of computation and communication). To improve the performance of the method, in this paper, we study an s-step variant of thick-restart Lanczos (TRLan) combined with an explicit external deflation (EED). The s-step method generates a set of s basis vectors at a time and reduces the communication costs of generating the basis vectors. We then design a specialized matrix powers kernel (MPK) that reduces both the communication and computational costs by taking advantage of the special properties of the deflation matrix. We conducted numerical experiments of the new TRLan eigensolver using synthetic matrices and matrices from electronic structure calculations. The performance results on the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC) demonstrate the potential of the specialized MPK to significantly reduce the execution time of the TRLan eigensolver. The speedups of up to $3.1\times$ and $5.3\times$ were obtained in our sequential and parallel runs, respectively.**

I. Introduction

The Lanczos method [1] is based on the Krylov subspace projection and is effective for computing a few exterior eigenvalues and their corresponding eigenvectors of a large Hermitian matrix. However, the computational cost of the method grows quickly with the number of the eigenvalues to be computed. As a result, when a large number of eigenvalues is needed, the method could become expensive. Moreover, in order to maintain the numerical stability of computing many eigenvalues, the method must be carefully implemented.

To improve the performance of the Lanczos method for computing a large number of eigenpairs, in this paper, we first combine thick-restart Lanczos (TRLan) [14] with the explicit external deflation (EED) [2]. We show that EED provides additional flexibility to the solver and may greatly improve the solver performance for computing the eigenpairs to a user-specified accuracy.

We then develop an s-step variant [3], [4] of TRLan combined with EED to improve the performance of its main computational kernel (the generation of the projection subspace). The s-step variant uses the matrix powers kernel (MPK) that generates a set of s basis vectors at a time by multiplying an input vector with a sparse matrix plus a low-rank matrix s times. Thus, MPK has the potential to reduce the communication cost of generating the basis vectors by a factor of s. We also design a specialized MPK that takes advantage of the special properties of our low-rank matrix (i.e., they consist of the computed eigenvectors) and reduces both the communication and computational costs of generating the basis vectors, while maintaining the numerical stability under certain assumptions. On modern computers, the communication cost, which includes moving the data between the parallel processing units as well as between the levels of the local memory hierarchy, has become significantly more expensive compared with the computation. Consequently, reducing communication could significantly improve the performance.

The main contributions of the paper are: (1) we show the flexibility of EED to improve the performance of the Lanczos method while maintaining its numerical stability, (2) for the s-step variant to generate the set of basis vectors, we describe three different MPK algorithms (standard, blocking cover, and specialized), where the second algorithm reduces the communication cost of the standard MPK, while the third algorithm reduces both the communication and computational costs, and (3) we compare the performance and numerical stability of the three MPKs on the Cori supercomputer at the National Energy Scientific Computing Center (NERSC). The performance results demonstrate the potential of the s-step variant to significantly reduce the execution time of the TRLan eigensolver. The speedups of up to $3.1\times$ and $5.3\times$ were obtained in our sequential and parallel runs, respectively. Since EED can be easily integrated into many of the existing eigensolvers (including those for general eigenvalue problems), the current paper may have broader impacts beyond the Lanczos method.

The rest of the paper is organized as follows. After surveying the related work in Section II, we outline TRLan and EED in Sections III and IV, respectively. We then in Section V describe an s-step variant of TRLan combined with EED (s-step TRLan+EED). Finally, in Section VII, we study the performance of s-step TRLan+EED. Final remarks are in Section VIII. Throughout this paper, we denote the jth column of a matrix A by $\mathbf{a}_j$, while $A_{j_1:j_2}$ is the submatrix consisting of the j_1th through the j_2th columns of A, and $A_{i_1:i_2,j_1:j_2}$ is the submatrix consisting of the i_1th through the i_2th rows and the j_1th through the j_2th columns of A. The superscript H indicates the conjugate transpose.

II. Related Work

To compute a large number of eigenpairs of a sparse matrix, there are several other techniques including contour

1530-2075/19/$31.00 ©2019 IEEE
DOI 10.1109/IPDPS.2019.00057

integral [5], [6], and spectral slicing or polynomial filtering [7], [8]. Though we plan to compare the performance of our implementation with these techniques, the focus of the current paper is to demonstrate the effectiveness of the Lanczos method combined with EED to compute a large number of eigenvalues, and to improve its performance using the s-step variant.

Communication-avoiding kernels have been integrated into s-step Lanczos method to improve the performance of the eigensolver [3], [9]. These s-step variants aim to generate a set of s basis vectors at a time, potentially reducing the communication latency cost by a factor of s [4]. The MPK for a general sparse-plus-low-rank matrix vector operation can be derived from the blocking cover algorithm [10], [11]. We compare the performance and stability of our implementation with those of the blocking cover algorithm. To the best of our knowledge, this is the first performance study and practical use of the MPK based on the blocking cover algorithm.

III. Thick-restart Lanczos (TRLan)

The Lanczos method [1] is a Krylov subspace projection method for computing a few eigenvalues $\bar{\lambda}_j$ and their corresponding eigenvectors $\bar{\mathbf{u}}_j$ of a Hermitian matrix A:

$$A\bar{\mathbf{u}}_j = \bar{\lambda}_j \bar{\mathbf{u}}_j,$$

where $\bar{\lambda}_1 \leq \bar{\lambda}_2 \leq \cdots \leq \bar{\lambda}_n$. Given a properly chosen starting vector $\mathbf{q}$, the method computes the orthonormal basis vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{j+1}$ of a Krylov subspace $\mathcal{K}_{j+1}(A, \mathbf{q}) \equiv \text{span}\{\mathbf{q}, A\mathbf{q}, \ldots, A^j\mathbf{q}\}$. These basis vectors satisfy the relation

$$AQ_j = Q_j T_j + \beta_j \mathbf{q}_{j+1} \mathbf{e}_j^H, \tag{1}$$

where $Q_j = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j]$, $\beta_j = \mathbf{q}_{j+1}^H A \mathbf{q}_j$, $\mathbf{e}_j$ is the jth column of the j-dimensional identity matrix, and $T_j = Q_j^H A Q_j$ is the $j \times j$ Rayleigh-Ritz projection of A onto $\mathcal{K}_j(A, \mathbf{q})$.

An approximate eigenpair $(\lambda, \mathbf{u} := Q_j \mathbf{x})$, referred to as a Ritz value and a Ritz vector, of A can be computed from an eigenpair $(\lambda, \mathbf{x})$ of T_j. Its residual norm satisfies

$$\|A\mathbf{u} - \lambda \mathbf{u}\|_2 = |\beta_j| \cdot |\mathbf{x}(j)|. \tag{2}$$

It is well known that Ritz values converge to exterior eigenvalues of A with a subspace dimension j that is much smaller than the dimension n of A [2], [12].

In the Lanczos method, the projected matrix T_j in (1) is symmetric tridiagonal of the form

$$T_j = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \alpha_{j-1} & \beta_{j-1} \\ & & \beta_{j-1} & \alpha_j \end{pmatrix}.$$

Hence, in exact arithmetic, the new basis vector $\mathbf{q}_{j+1}$ can be computed by orthonormalizing the vector $A\mathbf{q}_j$ against two preceding basis vectors, $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$:

$$\beta_j \mathbf{q}_{j+1} = A\mathbf{q}_j - \alpha_j \mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}. \tag{3}$$

This feature is commonly known as the *three-term recurrence*.

However, in finite precision arithmetics, when the new basis vector $\mathbf{q}_{j+1}$ is computed by (3), orthogonality among the basis vectors is lost even after a small number of iterations. To maintain the orthogonality, we reorthogonalize the new basis vector $\mathbf{q}_{j+1}$ against all the previous vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j$ using the classical Gram-Schmidt (CGS) procedure [12],

$$\widehat{\beta}_j \mathbf{q}_{j+1} := \mathbf{q}_{j+1} - Q_j (Q_j^H \mathbf{q}_{j+1}). \tag{4}$$

As the subspace dimension j grows, this procedure becomes expensive in terms of both computation and storage. To reduce the costs of computing a large subspace, we restart the iteration after a fixed number m of basis vectors are computed.

To maintain the convergence rate of the Ritz pairs after the restart, we explicitly keep some of the Ritz vectors computed at restart. For instance, when computing the smallest eigenvalues of A in our experiments, we keep the smallest converged Ritz values and one largest Ritz value (using the default restarting scheme 1 of TRLan [13]). To distinguish the basis vectors computed before and after the restart, we use $\widehat{Q}_k$ to denote the restarted basis vectors

$$\widehat{Q}_k = Q_m[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell, \mathbf{x}_m],$$

where $\mathbf{x}_i$ is the eigenvector corresponding to the ith smallest eigenvalue of T_m, and ℓ is the number of converged Ritz values, hence $k = \ell + 1$.

Then, to recover the three-term recurrence, we set the last Ritz vector $\mathbf{q}_{m+1}$ as the new $(k+1)$th basis vector $\widehat{\mathbf{q}}_{k+1}$, and compute the $(k+2)$th basis vector $\widehat{\mathbf{q}}_{k+2}$ by explicitly orthonormalizing $A\widehat{\mathbf{q}}_{k+1}$ against the previous $k+1$ basis vectors. Hence, at the jth iteration after the restart, the new basis vector $\widehat{\mathbf{q}}_{k+j+1}$ satisfies the relation:

$$A\widehat{Q}_{k+j} = \widehat{Q}_{k+j}\widehat{T}_{k+j} + \widehat{\beta}_{k+j}\widehat{\mathbf{q}}_{k+j+1}\mathbf{e}_{k+j}^H, \tag{5}$$

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

$$\widehat{T}_{k+j} = \begin{pmatrix} D_k & \beta_m \mathbf{s} & & & \\ \beta_m \mathbf{s}^H & \widehat{\alpha}_{k+1} & \widehat{\beta}_{k+1} & & \\ & \widehat{\beta}_{k+1} & \widehat{\alpha}_{k+2} & \ddots & \\ & & \ddots & \ddots & \widehat{\beta}_{k+j-1} \\ & & & \widehat{\beta}_{k+j-1} & \widehat{\alpha}_{k+j} \end{pmatrix}, \tag{6}$$

with D_k being the $k \times k$ diagonal matrix consisting of the kept Ritz values, and $\mathbf{s} = [\mathbf{x}_1, \ldots, \mathbf{x}_\ell, \mathbf{x}_m]^H \mathbf{e}_m$. A detailed description of TRLan can be found in [14].

IV. Explicit External Deflation (EED)

In order to maintain the numerical robustness of computing a large number of eigenpairs, TRLan implements a technique called *internal deflation* or *locking* [2]. Specifically, when a Ritz pair has converged to a small residual norm $\|A\mathbf{u} - \lambda\mathbf{u}\|_2 < \epsilon\|A\|_2$, TRLan assumes that the Ritz pair has converged to the machine precision and set its residual norm to be zero. Then, TRLan deflates these Ritz pairs by orthogonalizing the new basis vector against the locked Ritz

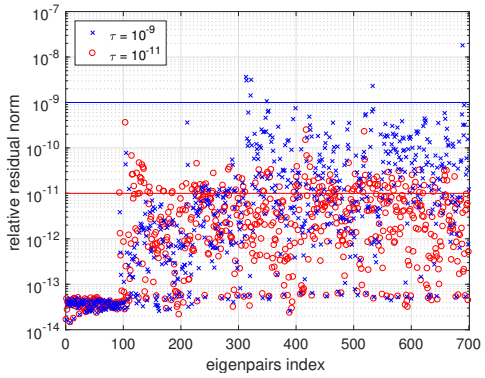


Fig. 1. Relative residual norms $\|A\mathbf{u} - \lambda\mathbf{u}\|_2/\|A\|_2$ of the computed eigenpairs. We used TRLan+EED to compute 100 eigenpairs of `SiH4` matrix ($n = 5041$) at a time (see Section VII for experiment setups).

vectors so that TRLan will not start computing these locked Ritz vectors again.

EED [2] provides an additional flexibility over locking for computing many eigenpairs. To integrate EED into TRLan, we incrementally compute a fixed number of Ritz pairs by running TRLan with the following shifted matrix:

$$A_d := A + \alpha U_d U_d^H,$$

where U_d contains the d converged Ritz vectors and α is a properly chosen shift. Hence, the converged eigenvalues are shifted away from the zero, and the next set of the desired eigenvalues becomes the exterior eigenvalues of the matrix A_d. It has been argued that even though the deflated Ritz pairs are computed only to a specified accuracy, EED can compute all the Ritz pairs to the specified accuracy [15]. Figure 1 shows that the relative residual norms of the eigenpairs computed by TRLan combined with EED (TRLan+EED) are kept below the specified threshold.

While TRLan only internally deflates the Ritz pairs that have converged to the machine precision, EED provides an additional flexibility to deflate the Ritz pairs as soon as they have converged to the desired accuracy. Thus, TRLan+EED may require a fewer number of iterations to compute all the Ritz pairs to a specified accuracy, while TRLan may unnecessarily compute some of the Ritz pairs to much higher accuracies. In addition, compared with TRLan+EED, TRLan often requires a much larger projection subspace to compute the eigenpairs.

Since the convergence behavior of TRLan and TRLan+EED depend greatly on the projection subspace dimension, it is difficult to conduct a fair comparison. To provide some idea of their performance, Figure 2 compares the two algorithms using different schemes to adjust the subspace dimension. It shows that under such setups, TRLan either performs more iterations using about the same amount of memory ($m = d + 200$), or requires much larger projection subspaces in order to obtain a convergence rate similar to TRLan+EED. Section VII-A describes our experimental setups in more details.

	#restart	#ops	# locked	time (s)
TRLan, fixed $m = 1400$	5	3,936	695	33.5
TRLan, adapt $m = 2(d + 100)$	38	11,327	656	52.0
TRLan, adapt $m = d + 200$	>86	>100,932	–	> 515.1
TRLan+EED, $m = 200$	122	12,257	556	25.8

(a) `SiH4` matrix ($n = 5,041$).

	#restart	#ops	# locked	time (s)
TRLan, fixed $m = 1400$	14	10,233	614	2,136.0
TRLan, adapt $m = 2(d + 100)$	77	22,356	613	3,158.7
TRLan, adapt $m = d + 200$	–	–	–	>7,200.0
TRLan+EED, $m = 200$	264	26,896	552	2,114.1

(b) `Si34H36` matrix ($n = 97,569$).

Fig. 2. Performance comparison of TRLan and TRLan+EED for computing 700 eigenpairs of `SiH4` matrix with $n = 5041$. With TRLan+EED, we computed 100 eigenpairs at a time with the projection subspace dimension set to be $m = 200$. With TRLan, we tested three different approaches to adjust the projection subspace: 1) we use a fixed projection dimension of $m = 1400$ for computing 700 eigenpairs at once; 2) we increase the projection dimension to $m = 2(d + 100)$ as the set of 100 eigenpairs have converged, where d is the number of converged eigenvalues; and 3) we increase the projection dimension to $m = d + 200$ as the set of 100 eigenpairs have converged. The last approach for TRLan requires about the same amount of memory as TRLan+EED, while the first two use larger projection subspaces to improve the convergence.

V. s-Step TRLan Combined with EED

The s-step variant [3] was developed to improve the performance of the classical Lanczos method by reducing its communication cost. In this paper, we develop s-step variants of TRLan+EED.

The s-step TRLan+EED relies on three main kernels:

1) Matrix powers kernel generates a set of s Krylov basis vectors by multiplying the sparse-plus-low-rank matrix $A + \alpha U_d U_d^H$ with a starting vector $\mathbf{q}_i$, i.e., $\mathbf{p}_{i+j} := (A + \alpha U_d U_d^H)\mathbf{q}_{i+j-1}$ for $j = 1, 2, \ldots, s$.
2) Block orthogonalization (blockOrtho) kernel orthogonalizes the set of the new basis vectors generated by MPK against the previously-generated orthonormal basis vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_i$. In our experiments, we used the block variant of the classical Gram-Schmidt procedure [12].
3) Tall-skinny QR (TSQR) factorization orthonormalizes the new basis vectors $\mathbf{p}_{i+1}, \mathbf{p}_{i+2}, \ldots, \mathbf{p}_{i+s}$ by computing its QR factorization. For our experiments, we use the Cholesky QR (CholQR) factorization [16].

Figure 3 shows the pseudocode of the resulting algorithm. To maintain the orthogonality among the basis vectors, we perform reorthogonalization as needed.

Compared with the standard TRLan+EED, the s-step variant has the potential to reduce the communication cost by a factor of s. For instance, on each process, blockOrtho and TSQR can orthogonalize the set of s vectors using level 3 BLAS operations to perform most of its local computation. On a distributed-memory computer, we distribute the matrix among the processes in 1D block row fashion. Then, both blockOrtho and TSQR only require one single global-reduce among the processes (see Figure 4 for an illustration of TSQR). In contrast, the standard algorithm orthogonalizes one vector at

set $\mathbf{q}_1 = \mathbf{q}/\|\mathbf{q}\|_2$, $k = 0$.
for $j = 1, 2, 3, \ldots$
1. Initialization.
   a. $\mathbf{p} := (A + \alpha U_d U_d^H)\mathbf{q}_{k+1}$
   b. $\alpha_{k+1} := \mathbf{q}_{k+1}^H \mathbf{p}$
   c. $\mathbf{p} := \mathbf{p} - \alpha_{k+1}\mathbf{q}_{k+1} - \sum_{i=1}^{k} \beta_i \mathbf{q}_i$
   d. $\beta_{k+1} := \|\mathbf{p}\|_2$
   e. $\mathbf{q}_{k+2} := \mathbf{p}/\beta_{k+1}$
2. The j-th restart-loop.
   for $i = k+2 : s : m$
   a. Starting vector $\mathbf{p}_i = \mathbf{q}_i$.
   b. Matrix Powers Kernel:
      for $\ell = i, i+1, \ldots, i+s-1$
         $\mathbf{p}_{\ell+1} := (A + \alpha U_d U_d^H)\mathbf{p}_\ell$
      end for
   c. Block three-term orthogonalization:
      $R_{i-s:i,\ i+1:i+s} := Q_{i-s:i}^H P_{i+1:i+s}$
      $P_{i+1:i+s} := P_{i+1:i+s} - Q_{i-s:i} R_{i-s:i,\ i+1:i+s}$
   d. Tall-skinny Cholesky QR factorization:
      $B := P_{i+1:i+s}^H P_{i+1:i+s}$
      $R_{i+1:i+s,\ i+1:i+s} := \text{chol}(B)$
      $Q_{i+1:i+s} := P_{i+1:i+s} R_{i+1:i+s, i+1:i+s}^{-1}$
   e. Reorthogonalize $Q_{i+1:i+s}$ if necessary:
      Classical Gram Schmidt followed by Cholesky QR.
   f. Update the projected matrix T_m:
      see, e.g., [4, Sec. 4.2.2].
   end for
3. The j-th restart.
   a. compute all eigenpairs of T_m and the corresponding residual norms for Ritz pairs by (2).
   b. if stopping criteria is satisfied then
   c. compute desired Ritz vectors and exit.
   d. else restart:
   e. update k (see [14], [13]).
   f. select k Ritz values $\lambda_1, \ldots, \lambda_k$ of interest, and compute their Ritz vectors $\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}$.
   g. set $\alpha_i = \lambda_i$ and $\beta_i = \|A\mathbf{q}_i - \lambda_i \mathbf{q}_i\|_2$ by (2), for $i = 1, \ldots, k$,
   h. set $\mathbf{q}_{k+1} = \mathbf{q}_{m+1}$.
   i. end if
end for

Fig. 3. Pseudocode of s-step TRLan + EED.

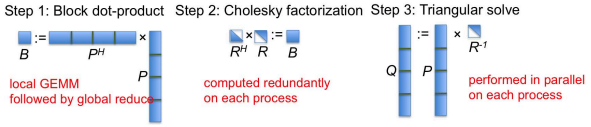


Fig. 4. Illustration of Cholesky QR factorization.

a time using level 2 BLAS local computation and two global-reduces.

VI. Matrix Powers Kernels for s-step TRLan+EED

Now we describe our main contribution of the paper, designing the MPK to apply s sparse-plus-low-rank matrix operations on a starting vector $\mathbf{p}_0$. Namely, for $j = 1, 2, \ldots, s$, MPK generates the vectors

$$\mathbf{p}_j \quad := \quad (A + \alpha U_d U_d^H)^j \mathbf{p}_0. \tag{7}$$

The *standard MPK* computes the s basis vectors by the straightforward recursion

$$\mathbf{p}_j := A\mathbf{p}_{j-1} + \alpha U_d (U_d^H \mathbf{p}_{j-1}).$$

1. dot-products
   $\mathbf{b}_0 := \alpha U_d^H \mathbf{p}_0$

2. local computation
   for $j = 1, 2, \ldots, s-1$ do
      $\mathbf{b}_j := W_{j-1}\mathbf{b}_0$
   end for

3. local matrix-matrix multiplication
   $[\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{s-1}] := U_d[\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{s-1}]$

4. MPK with a sparse matrix A
   for $j = 1, 2, \ldots, s$ do
      $\mathbf{p}_j := A\mathbf{p}_{j-1} + \mathbf{c}_{j-1}$
   end for

Fig. 5. Specialized MPK to generate the s basis vectors $\mathbf{p}_j$ for $j = 1, 2, \ldots, s$ in (7), using $W_j = (\Lambda_d + \alpha I)^j$ and assuming (8).

This requires a global-reduce at each step of MPK.

A. Specialized MPK

To reduce the computational and communication costs of the standard MPK, we assume the following properties of the Ritz vectors U_d,

$$AU_d = U_d\Lambda_d \quad \text{and} \quad U_d^H U_d = I, \tag{8}$$

where $\Lambda_d = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$ has the computed d eigenvalues on the diagonal, and $U_d = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d]$ consists of the corresponding eigenvectors.

By (8), we have

$$U_d^H (A + \alpha U_d U_d^H)^j \quad = \quad (\Lambda_d + \alpha I)^j U_d^H. \tag{9}$$

As a result, the jth vector $\mathbf{p}_j$ of (7) satisfies

$$\begin{aligned} \mathbf{p}_j \quad &\equiv \quad (A + \alpha U_d U_d^H)^j \mathbf{p}_0 \\ &= \quad (A + \alpha U_d U_d^H)(A + \alpha U_d U_d^H)^{j-1} \mathbf{p}_0 \\ &= \quad A(A + \alpha U_d U_d^H)^{j-1} \mathbf{p}_0 \\ &\quad\ + \alpha U_d [U_d^H (A + \alpha U_d U_d^H)^{j-1}] \mathbf{p}_0 \\ &= \quad A\mathbf{p}_{j-1} + \alpha U_d (\Lambda_d + \alpha I)^{j-1} U_d^H \mathbf{p}_0. \end{aligned} \tag{10}$$

This recursion allows us to design a *specialized MPK* to compute $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_s$ as illustrated in Figure 5.

Generating the monomial basis vectors based on (10) without orthogonalization can be numerically unstable because the generated basis vectors converge to the eigenvector corresponding to the dominant eigenvalue of A. To improve the numerical stability, we generate the Newton basis such that

$$\mathbf{p}_j := [(A - \lambda_j I) + U_d U_d^H]\mathbf{p}_{j-1},$$

where we use the Ritz values of the matrix A in a Leja ordering as the shifts λ_j [17]. Given a starting vector $\mathbf{p}_0$ and a set of shifts $\lambda_1, \lambda_2, \ldots, \lambda_s$, the specialized MPK algorithm can be extended to generate the Newton basis. Namely, we can extend (9) such that

$$U_d^H \prod_{k=1}^{j} [(A + \lambda_k I) + \alpha U_d U_d^H] \quad = \quad \prod_{k=1}^{j} [\Lambda_d + (\alpha + \lambda_k) I] U_d^H.$$

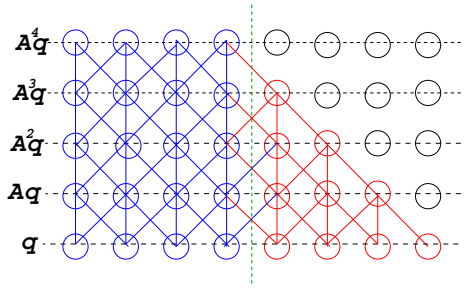


Fig. 6. Illustration of MPK for SpMV with a tridiagonal matrix. The blue circles show the local elements of the process to be computed, while the red circles are the ghost entries to be redundantly computed among the neighboring processes.

and hence, we have

$$\mathbf{p}_j \quad := \quad (A+\lambda_j I)\mathbf{p}_{j-1} + \alpha U_d W_{j-1} U_d^H \mathbf{p}_0, \qquad (11)$$

where

$$W_{j-1} = \prod_{k=1}^{j-1} [\Lambda_d + (\alpha+\lambda_k) I]. \qquad (12)$$

Given a new set of shifts at each restart, it requires only a small amount of computation to generate the auxiliary diagonal matrices W_{j-1}.

At the last step of MPK in Figure 5, the standard sparse-matrix vector multiplication (SpMV) requires a point-to-point communication to gather the required elements of each input vector $\mathbf{p}_{j-1}$ from the neighboring processes. It is possible to design an MPK that applies s SpMVs with a single communication latency cost [18]. Namely, each process first gathers the s-level ghost entries of the starting vector $\mathbf{q}_0$ from the neighboring processes (i.e., the nonlocal entries that are s edges away from the local entries in the adjacency graph of A). After this round of point-to-point communication, each process may independently apply SpMV s times without further communication (see Figure 6 for an illustration).

Though MPK reduces the communication latency to generate the s basis vectors by a factor of s, each jth step of MPK applies SpMV to the $(s-j+1)$-level ghost elements. This requires extra computation. In addition, depending on the surface-to-volume ratio of the matrix partitioning, the total communication volume may increase [18]. In many cases, this point-to-point communication is less significant compared with the all-reduce communication required for the deflation and orthogonalization (as we will show in Section VII).

It is also possible to implement an MPK with a local sparse matrix, which loads the local matrix into a fast memory only once without any overhead. However, its implementation needs to be carefully tuned on each hardware architecture. Thus, in this paper, we focus on reducing the communication associated with the orthogonalization and deflation, while using the standard algorithm for SpMV. A similar approach was used in our previous task-based implementations of the s-step and pipelined solvers [19].

	computation flop count	communication, intra volume	inter latency
1	$s \cdot \text{nnz}(A) + 2nds$	$s \cdot \text{nnz}(A) + 2nds$	$s+s$
2	$(2s-1) \cdot \text{nnz}(A) + 2nds$	$(2s-1) \cdot \text{nnz}(A) + 2nd$	$2+1$
3	$s \cdot \text{nnz}(A) + nd \cdot (s+1)$	$s \cdot \text{nnz}(A) + 2nd$	$1+1$

Fig. 7. Cost of three different MPK to generate a set of s basis vectors: 1) standard, 2) blocking cover, and 3) specialized. For communication of specialized and blocking cover algorithms, we assume U_d needs to be read into fast memory once for each GEMM operation for 2) and 3), while it needs to be read into fast memory for each GEMV operation for 1). We need one GEMV and one GEMM for 3). For inter-process latency, we assume one point-to-point communication for the set of s SpMVs, but in our implementation we perform $2s-1$ and s communications for 1) and 2), respectively.

B. Communication and Computation Costs

To generate the set of s basis vectors, the specialized MPK accesses U_d only twice. The standard MPK would access U_d once for generating each vector $\mathbf{p}_j$. On a distributed-memory computer, we distribute all the matrices (e.g., A, Q_j, and U_d) among the processes in a 1D block row format. Hence, the specialized MPK needs only one all-reduce to compute $\mathbf{b}_0$, while the standard MPK requires one all-reduce for generating each basis vector (Thus a total of s all-reduces for generating the s basis vectors).

In addition to reducing the communication cost, the specialized MPK takes advantage of the condition (8) and reduces the computational cost for applying the deflation. Specifically, this specialized algorithm requires only $\mathcal{O}(s(\text{nnz}(A)+n)+(s+1)nd+(s-1)d)$ floating-point operations (FLOPs), compared with $\mathcal{O}(s(\text{nnz}(A)+n)+2nds)$ FLOPs needed by the standard algorithm. Figure 7 summarizes the costs of the three MPK algorithms studied in this paper (the blocking cover algorithm will be discussed in Section VI-D).

C. Numerical Stability

The specialized MPK is derived based on the conditions (8) that assume the deflation eigenpairs are exact. In practice, the eigenpairs (Λ_d, U_d) are computed only to a specified accuracy, so that

$$AU_d = U_d\Lambda_d + E \quad \text{and} \quad U_d^H U_d = I + F, \qquad (13)$$

where E consists of the residual vectors of the computed eigenpairs, and F is the error in orthogonalization due to rounding error. The magnitude of E is determined by the stopping criteria in TRLan, and, assuming the orthogonality among the basis vectors are maintained, we have $\|F\|_2 = \mathcal{O}(\epsilon)$ where ϵ is the machine precision.

Due to conditions (13), the equality (9) now becomes

$$U_d^H (A+\alpha U_d U_d^H)^j \quad = \quad (\Lambda_d + \alpha I)^j U_d^H + E_j,$$

where $E_0 = 0$, and, for $j \geq 1$,

$$E_j = \sum_{i=0}^{j-1} (\Lambda_d+\alpha I)^i (E+\alpha U_d F)^H (A+\alpha U_d U_d^H)^{j-1-i}. \qquad (14)$$

As a result, the jth vector $\mathbf{p}_j$ in (10) now satisfies

$$\begin{aligned} \mathbf{p}_j \quad &\equiv \quad (A+\alpha U_d U_d^H)^j \mathbf{p}_0 \\ &= \quad A\mathbf{p}_{j-1} + \alpha U_d (\Lambda_d+\alpha I)^{j-1} U_d^H \mathbf{p}_0 + \alpha U_d E_{j-1} \mathbf{p}_0. \end{aligned}$$

In comparison, the jth vector by the specialized MPK, denoted by $\widetilde{\mathbf{p}}_j$, satisfies

$$\widetilde{\mathbf{p}}_j \equiv A\widetilde{\mathbf{p}}_{j-1} + \alpha U_d(\Lambda_d + \alpha I)^{j-1} U_d^H \mathbf{p}_0,$$

with $\widetilde{\mathbf{p}}_0 = \mathbf{p}_0$.

We thus obtain

$$\mathbf{d}_j = A\mathbf{d}_{j-1} + \alpha U_d E_{j-1}\mathbf{p}_0 \quad \text{with} \quad \mathbf{d}_j = \mathbf{p}_j - \widetilde{\mathbf{p}}_j.$$

Hence, the difference in the jth vectors between the standard and specialized MPK satisfies

$$\mathbf{d}_j = \alpha \sum_{i=0}^{j-2} A^i U_d E_{j-i-1}\mathbf{p}_0.$$

By reasonably assuming $|\alpha| \leq \|A\|_2$ and $\|U_d\|_2 \leq 2$, we have

$$\begin{aligned} \|\mathbf{d}_j\|_2 &\leq 2\alpha \left(\sum_{i=0}^{j-2} \|A\|_2^i \|E_{j-i-1}\|_2 \right) \\ &\leq 3j(j-1)\alpha(\|A\|_2 + \alpha)^{j-2} \cdot \eta + \mathcal{O}(\eta^2), \quad (15) \end{aligned}$$

where

$$\eta = \max\{\|E\|_2,\ \|A\|_2\|F\|_2\},$$

due to the facts that E_i is given by (14), and $\|\alpha F\|_2 \leq \|A\|_2\|F\|_2 \leq \eta$ and $\|\Lambda_d\|_2 \leq \|A\|_2 + \mathcal{O}(\eta)$ by (13).

We would like the error (15) in the specialized MPK to be in the order of $\mathcal{O}(\epsilon n(\|A\|_2+\alpha)^j)$; namely, the round-off errors in evaluating $\mathbf{p}_j$ by the standard MPK (7). This requires

$$\eta \leq \mathcal{O}\left(\epsilon\ n\ \alpha^{-1}(\|A\|_2 + \alpha)^2\right).$$

Such a condition can hold if the relative residual norms of the approximate eigenpairs satisfy

$$\frac{\|E\|_2}{\|A\|_2} \leq \tau \leq \frac{\epsilon\ n(\|A\|_2 + \alpha)^2}{\alpha\|A\|_2}, \quad (16)$$

where τ is the tolerance used for the residual norm of the computed eigenpairs. As we will show in our experiments, these requirements can be met in most of the practical cases.

D. Blocking Cover

The blocking cover algorithm [10], [11] can be adapted to implement the MPK that generates the basis vectors by multiplying a general sparse-plus-low-rank matrix $A + \alpha U_d U_d^H$ to a starting vector s times, where U_d is an $n \times d$ general matrix. Figure 8 shows the pseudocode of the algorithm. Similar to our specialized MPK in Section VI-A, this *blocking cover* MPK generates the s basis vectors through one all-reduce (Step 2 of Figure 8). However, to reduce the communication cost, this algorithm requires $2s-1$ SpMVs with the matrix A for generating the s basis vectors (Steps 1 and 5). In contrast, the standard and specialized MPK algorithms require s SpMVs. Figure 7 compares the costs of this algorithm with those of the standard and specialized MPK algorithms.

In addition, the blocking cover MPK algorithm requires $s-1$ sparse-matrix matrix multiplications (SpMMs) to compute the auxiliary matrices $X_j := U_d^H A^{j-1} U_d$ for $j = 1, \ldots, s-1$. Since the columns of U_d are the Ritz vectors of A, for the

```
1. MPK with a sparse matrix A
  for j = 1, 2, ..., s − 1 do
     p_j := A p_{j−1}
  end for

2. Block dot-product
  B := U_d^H · [p_0, p_1, ..., p_{s−1}]

3. local computation, O(ds^2) flops
  for j = 1, 2, ..., s do
     c_j := b_j
     for i = 1, 2, ..., j − 1 do
        c_j := c_j + X_i c_{j−i}
     end for
     c_j := α c_j
  end for

4. generate low-rank correction
  Y := U_d · [c_1, c_2, ..., c_s]

5. MPK with A to integrate low-rank correction
  for j = 1, 2, ..., s do
     p_j := A p_{j−1} + y_j
  end for
```

Fig. 8. Blocking cover MPK to compute $\mathbf{p}_j := (A + \alpha U_d U_d^H)^j \mathbf{p}_0$ with general sparse and dense matrices A and U_d, respectively, for $j = 1, 2, \ldots, s$, where $X_j := U_d^H A^{j-1} U_d$.

algorithm setup in our experiments, we avoided these extra SpMVs using the following assumption:

$$X_j := U_d^H A^{j-1} U_d = \Lambda_d^{j-1}. \quad (17)$$

In our experiments, we did not see significant numerical effects by making this assumption.

The assumption (17) is milder than that in (8). In particular, for the Ritz pairs computed by Lanczos, we have

$$AU_d = U_d\Lambda_d + \mathbf{q}\mathbf{t}^H + E,$$

where $\mathbf{q}$ is numerically orthonormal to U_d (i.e., $\|\mathbf{q}\|_2 \approx 1$ and $U_d^H\mathbf{q} \approx 0$), and E is the numerical error in Lanczos satisfying $\|E\|_2 = \mathcal{O}(\|A\|_2\epsilon)$; see, e.g., [20]. Assuming full reorthogonalization is applied in the computation such that $\|U_d^H U_d - I\|_2 = \mathcal{O}(\varepsilon)$ and $\|U_d^H\mathbf{q}\|_2 = \mathcal{O}(\epsilon)$, and the eigenpairs are computed to at least half machine precision, i.e., $\|\mathbf{t}\|_2 \leq \mathcal{O}(\sqrt{\epsilon}\|A\|_2)$, we can derive by induction that $\|U_d^H A^{j-1}\mathbf{q}\|_2 \leq \mathcal{O}(\sqrt{\epsilon}\|A\|_2^{j-1})$ and $\|U_d^H A^{j-1} U_d - \Lambda_d^{j-1}\|_2 = \mathcal{O}(\epsilon\|A\|_2^{j-1})$. Hence, the error $\|X_j - \Lambda_d^{j-1}\|_2$ is of the same order as the numerical error for evaluating $X_j := U_d^H A^{j-1} U_d$, explicitly. The stability of the blocking cover MPK, in comparison with the specialized MPK, will be illustrated in Section VII-C.

VII. Experimental Results

We now study the performance of s-step TRLan+EED. After describing our experimental setups in Section VII-A, we first present in Section VII-B the performance of the main BLAS kernel used to implement our orthogonalization and deflation procedures. We then compare the sequential performance of the s-step TRLan-EED using the three different MPKs on one process, for synthetic matrices and for matrices

from electronic structure calculations. in Sections VII-C and VII-D, respectively. Finally, in Section VII-E, we examine the distributed-memory performance of the s-step TRLan-EED.

A. Experimental Setups

We conducted all of our experiments on the Haswell nodes of the Cori supercomputer at NERSC. A Haswell node has two 16-core Intel Xeon E5-2698 v3 Haswell CPUs at 2.3 GHz, and 128 GB of main memory with 40 MB of Smart Cache. The nodes are connected through the Cray Aries interconnect with Dragonfly topology. We compiled our code using Cori's compiler wrapper `cc` for the Intel C compiler `icc` version `18.0.1 20171018`.

For all of our experiments, unless specified otherwise, we performed CholQR twice to maintain the orthogonality of the basis vectors, and considered the computed eigenpairs to have converged when their relative residual norms became less than $\tau = 10^{-11}$. In addition, we computed 100 eigenpairs at a time and set the shift α for EED using the computed eigenvalues from the previous run of TRLan as $\alpha = \lambda_d + \frac{\lambda_n - \lambda_d}{2}$, where λ_d is the largest eigenvalue among the smallest converged eigenvalues and λ_n is the largest converged eigenvalue (see Figure 9 for an illustration).

Fig. 9. Illustration of converged eigenvalues λ_j and the shift α for EED.

B. Kernel Performance

The s-step TRLan+EED reduces the computational and communication costs of two main kernels: the CGS orthogonalization in (4) and the deflation operation in (7). The standard TRLan+EED uses level 2 BLAS matrix-vector multiply `GEMV` on each process and generates one basis vector at a time. The s-step TRLan+EED aims to orthonormalize and deflate a set of s basis vectors at a time, replacing the `GEMV` operations with the level 3 BLAS matrix-matrix multiply `GEMM` operations on each process. The level 2 BLAS performance is typically limited by the memory bandwidth on the process because the matrix elements in the fast memory cannot be reused during the consecutive `GEMV` operations (i.e., 2 FLOPs per data read). On a distributed-memory computer, the level 2 BLAS operation $Q_j^H \mathbf{q}_{j+1}$ or $U_d^H \mathbf{p}_0$ require a global-reduce among all the processes. The level 3 BLAS increases the data reuse (i.e., $2s$ FLOPs per data read). In addition, on a distributed-memory computer, each BLAS-3 operation needs only one global-reduce, potentially reducing the communication latency cost by a factor of s.

Figures 10 and 11 show the performance of these BLAS kernels, where we multiply an m-by-60 matrix U_{60} with a set of s basis vectors P. We see that higher performance is obtained using s basis vectors at a time.

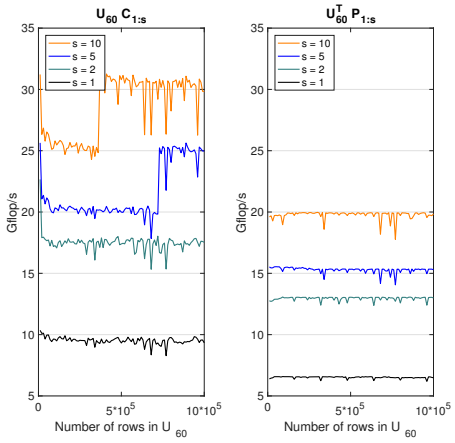


Fig. 10. Sequential performance of `dgemm` with $n = 60$.

	process count				
m	1	16	32	64	128
2×10^5	19.9	128.4	458.4	480.4	868.3
4×10^5	19.9	127.5	465.7	486.2	937.7
6×10^5	19.9	239.0	468.4	495.2	967.9

Fig. 11. Parallel performance (gigaFLOP/s) of `dgemm` with $n = 60$ and $s = 10$.

C. Synthetic Diagonal Matrices

We now study the performance of s-step TRLan+EED using the synthetic diagonal matrices $A_2 = \text{diag}(1^2, 2^2, \ldots, n^2)$ (which were previously used to study the performance of TRLan [9]). In Figure 12, we examine the effects of the residual norms of the computed eigenpairs on the stability of the specialized and blocking cover (marked as "general" in figures) MPKs. The s-step TRLan+EED failed using the specialized MPK with $\tau > 10^{-8}$ (where $\epsilon n \|A + \alpha U U^H\|_2 \approx 5.6 \cdot 10^{-8}$ for (16)) while the blocking cover MPK continues to converge with a larger value of τ. Namely, the solid and dashed lines overlap on each other for $\tau = 10^{-9}$ and 10^{-8} (blue and red), while only the dashed line converged for $\tau = 10^{-7}$ and 10^{-6} (green and black). We note that with $n = 1000$, the requested accuracy $\tau = 10^{-8}$ for the relative residual norm is relatively low (i.e., $\|A\mathbf{u} - \lambda\mathbf{u}\|_2 \leq 10^{-2}$).

In Figure 13, we compare the performance of s-step TRLan+EED using the standard, specialized, and blocking cover MPKs to generate the Krylov vectors. For computing the first set of eigenpairs, the blocking cover and specialized MPKs already reduced the execution time by replacing the standard orthogonalization kernel with blockOrtho and TSQR. For computing the additional eigenpairs, since this is a diagonal matrix, "MatOp" in the figure is dominated by the time for the deflation. We can see that the deflation operation becomes more expensive and dominates the iteration time as more eigenpairs are computed. The blocking cover MPK can reduce the time needed for the deflation by avoiding some data movement, while the specialized MPK can further reduce the execution time by taking advantage of the special properties

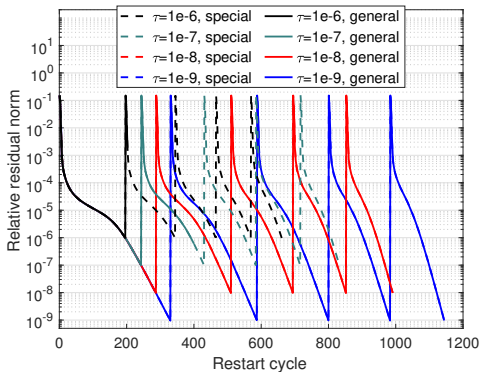


Fig. 12. Convergence history of s-step TRLan+EED to compute the five smallest eigenvalues (one eigenvalue at a time) of the diagonal matrix $A_2 = \text{diag}(1^2, 2^2, \ldots, n^2)$ with $n = 1000$, $m = 30$, $\alpha = 5000^2$, $s = 5$, and different values of τ.

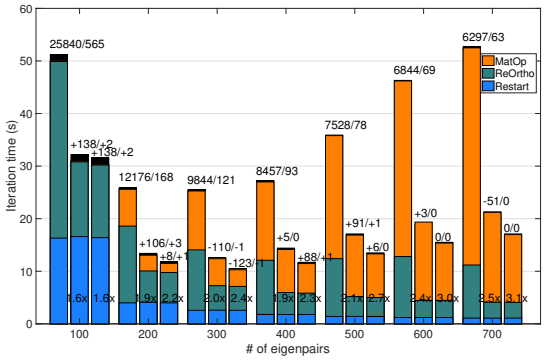


Fig. 13. Performance of s-step TRLan+EED for $A_2 = \text{diag}(1^2, 2^2, \ldots, n^2)$ to compute $100, 200, \ldots, 700$ eigenpairs. The left, middle, and right bars show the time needed to compute the next 100 eigenpairs using the standard, blocking cover and specialized MPKs, respectively. The numbers on top of the left bar show the number of restarts and the number of matrix operations needed for the convergence using the standard MPK, while the numbers on top of the middle and right bars show the differences in the restart and operation counts using the blocking cover and specialized MPKs, compared with the standard MPK, respectively. We used $n = 10000$, $m = 200$, $\alpha = 5000^2$, and $s = 10$, and computed 100 eigenpairs at a time.

of the matrices and avoiding some computation.

In Figure 14, we study the effect of the step size s on the performance of s-step TRLan+EED with the specialized MPK. A larger value of s improves the performance, but increases the potential of numerical instability. For the rest of the paper, we use $s = 5$ unless otherwise specified.

D. Matrices for Density Functional Theory Calculation

We now study the performance of s-step TRLan+EED using the matrices from the density functional theory for electronic structure calculations. These matrices are from the PARSEC matrix collection downloaded from the SuiteSparse Matrix Collection. As shown in the left plot of Figure 15, the eigenvalues of the PARSEC matrices are closely clustered. In order to avoid missing the eigenvalues, we generated the Newton basis

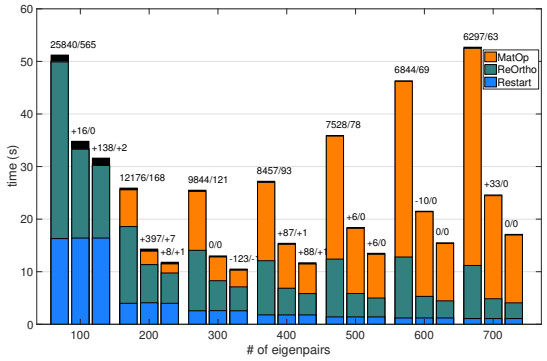


Fig. 14. Performance of s-step TRLan+EED combined with specialized MPK for $A_2 = \text{diag}(1^2, 2^2, \ldots, n^2)$ with $n = 100$, $m = 30$, and different values of $s = 1$, 5, or 10 (left, middle, and right bars, respectively).

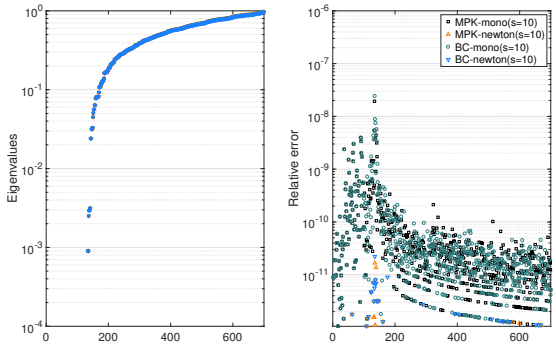


Fig. 15. Computed eigenvalues of `Si34H36` matrix from PARSEC collection (i.e., $n = 97,569$ and $nnz/n \approx 53$).

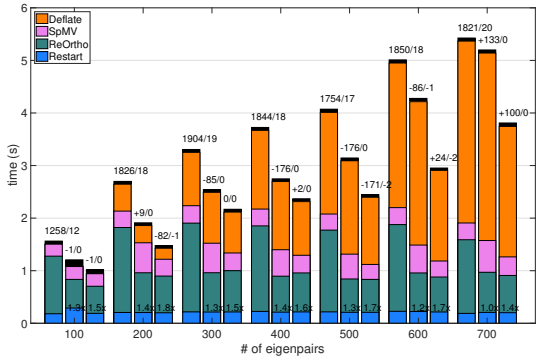


Fig. 16. Performance of TRLan+EED with the standard, blocking cover, and specialized MPKs (left, middle, and right bars in each set) for `SiH4` matrix from PARSEC collection (i.e., $n = 5,041$, $nnz/n \approx 34$, and $s = 5$).

using the Ritz values from the first restart (the first restart loop is based on the standard TRLan), and performed CholQR three times. The s-step TRLan+EED missed some eigenvalues with monomial basis.

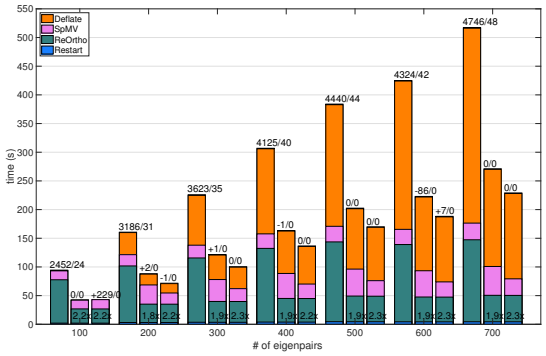


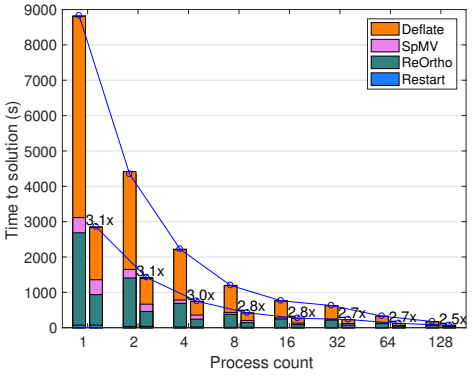
Fig. 17. Performance of s-step TRLan+EED with the standard, blocking cover, and specialized MPKs (left, middle, and right bars in each set) for `Si34H36` matrix from PARSEC collection (i.e., $n = 97,569$, $nnz/n \approx 53$, and $s = 5$).

Figures 16 and 17 compare the performance of s-step TRLan+EED using three different MPKs. We can see that the performance is improved by avoiding both the computation and communication, especially for large matrices. For large matrices, the performance was improved more by avoiding data movement (i.e., the blocking cover MPK obtained significant improvement over the standard MPK, while the specialized MPK obtained only a small improvement over the blocking cover MPK). On the other hand, for small matrices the performance was improved more by avoiding computation (i.e., the improvement by the blocking cover MPK was often smaller than that by the specialized MPK). For instance, when the matrix is small enough for all the data to fit in the fast cache, the effects of reducing the data traffic among the local memory hierarchy becomes less significant. Overall, the specialized MPK obtained the speedups of up to $2.3\times$, and the speedups were increased as more eigenvalues were computed.

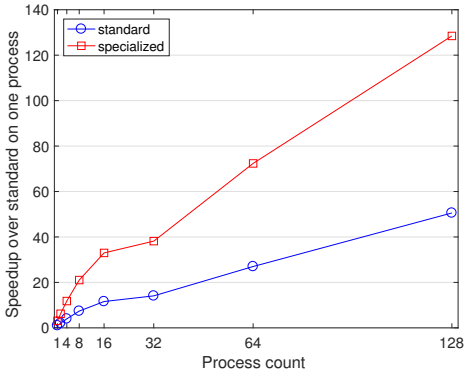
E. Parallel Performance

We now compare the performance of our distributed-memory implementation of s-step TRLan+EED using the standard and specialized MPK. Figure 18 shows the strong parallel scaling results with the `Si87H76` matrix from the PARSEC matrix collection. Though the specialized MPK improved the performance of s-step TRLan+EED, its speedup lowered as we increased the process count. One reason is that, using the specialized MPK, the sequential parts of the algorithm (e.g., restart and the sequential part of MPK) become significant, since the iteration time was reduced by optimizing the main computational kernel. We observed similar trends with the synthetic diagonal matrix in Figure 19.

To verify the numerical stability of our implementation, Figure 20 shows the relative residual norms of the computed eigenpairs. They were all below the specified threshold, but those by the specialized MPK seem to have larger values.



(a) Time breakdown.



(b) Speedup over standard MPK on one process.

Fig. 18. Strong-parallel scaling of s-step TRLan+EED to compute 100 eigenvalues of `Si87H76` at a time using standard and specialized MPK with $n = 240,369$, $m = 200$, $s = 5$.

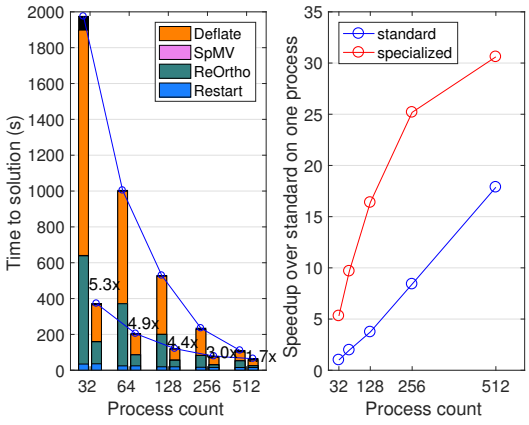


Fig. 19. Performance of s-step TRLan+EED to compute 100 eigenvalues of $A_1 = \mathrm{diag}(1, 2, \ldots, n)$ at a time using standard and specialized MPK with $n = 320,000$, $m = 200$, $s = 5$.

VIII. Conclusion

The thick-restart Lanczos combined with the explicit external deflation (TRLan+EED) is an efficient method for

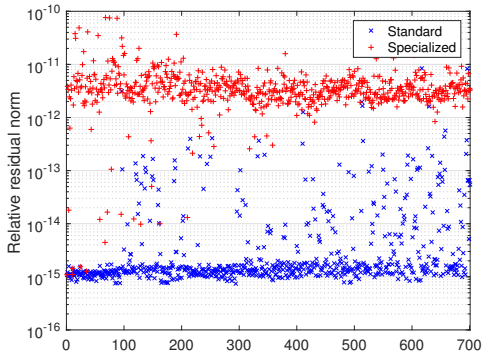


Fig. 20. Relative residual norms $\|A\mathbf{u}-\lambda\mathbf{u}\|_2/\|A\|_2$ of the computed eigenpairs for `Si87H76` using the tolerance of $\tau = 10^{-11}$, where $\|A\|_2 \approx 42.9$.

computing a large number of eigenvalues of a Hermitian matrix. To improve the performance of TRLan+EED, in this paper, we studied an s-step variant that aims to reduce both the communication and computational costs of its main computational kernel (i.e., generation of the subspace projection basis). Our experimental results on the Cori supercomputer at NERSC demonstrated the potential of these techniques. We are conducting experiments with larger matrices, where the inter-process communication can become more significant than our current sequential bottleneck on a larger number of processes, thus increasing the benefit of avoiding the inter-process communication.

Both the standard and s-step variants of TRLan+EED have the potential of missing eigenpairs, especially when the eigenvalues are closely clustered. We are looking at strategies to avoid missing eigenvalues (e.g., computing the extra eigenvalues at each run, then using the eigenvectors associated with the extra eigenvalues as the starting vectors for the next run).

We are also looking at the potential numerical issues. One source of instability comes from TRLan and s-step Lanczos themselves. The numerical behavior of TRLan has been studied in [14], while stability analysis in [21] has shown the s-step methods have a similar behavior to that of classical Lanczos (assuming a bound on the condition number of the s-step Krylov subspace). Both results indicate that orthogonality of the basis vectors is crucial. Though we did not encounter such issues in this paper, we are looking to integrate other orthogonalization procedures including the singular value and tall-skinny QR factorization [16], [22].

Another source of instability is due to EED using inexact eigenvectors U_d. When U_d is a single vector, Parlett [2, Chap. 5.1] shows that the change in a distant eigenpair caused by deflation of U_d is the same as the error of U_d along that eigenvector, and it will not be significant. Although this result does not immediately extend to deflating multiple vectors, the similar numerical behavior was observed in practice [15]. We would like to conduct further theoretical analysis to understand such behavior.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, the U.S. Department of Energy Office of Science under Award Numbers DE-FG0213ER26137 and DE-SC0010042, and the U.S. National Science Foundation under Awards 1339822, DMS-1522697, and CCF-1527091.

REFERENCES

[1] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Nat. Bur. Stand. 45 (1950) 255–281.
[2] B. N. Parlett, The symmetric eigenvalue problem, Classics in Applied Mathematics. SIAM, Philadelphia, PA, 1998.
[3] S. Kim, A. T. Chronopoulos, A class of Lanczos-like algorithms implemented on parallel computers, Parallel Computing 17 (1991) 763–778.
[4] M. Hoemmen, Communication-avoiding Krylov subspace methods, Ph.D. thesis, EECS Department, University of California, Berkeley (2010).
[5] T. Sakurai, H. Sugiura, A projection method for generalized eigenvalue problems, J. Comput. Appl. Math 159 (2003) 119–128.
[6] E. Polizzi, Density-matrix-based algorithms for solving eigenvalue problems, Phys. Rev. B 79 (2009) 115–112.
[7] C. Bekas, E. Kokiopoulou, Y. Saad, Computation of large invariant subspaces using polynomial filtered Lanczos iterations with applications in density functional theory, SIAM J. Matrix Anal. Appl. 30 (2008) 397–418.
[8] R. Li, Y. Xi, E. Vecharynski, C. Yang, Y. Saad, A Thick-Restart Lanczos algorithm with polynomial filtering for Hermitian eigenvalue problems, SIAM J. Sci. Comput. 38 (2016) A2512–A2534.
[9] I. Yamazaki, K. Wu, A communication-avoiding thick-restart Lanczos method on a distributed-memory system, in: Workshop on Algorithms and Programming Tools for next-generation high-performance scientific and software (HPCC), 2011, pp. 345–354.
[10] C. Leiserson, S. Rao, S. Toledo, Efficient out-of-core algorithms for linear relaxation using blocking covers, J. Comput. Syst. Sci. Int. 54 (1997) 332–344.
[11] N. Knight, E. Carson, J. Demmel, Exploiting data sparsity in parallel matrix powers computations, in: R. Wyrzykowski et al. (Eds.): PPAM 2013, Part I, LNCS 8384, pp. 1525, 2014.
[12] Y. Saad, Numerical Methods for Large Eigenvalue Problems, revised edition Edition, SIAM, Philadelphia, 2011.
[13] K. Wu, H. Simon, TRLan user guide, Tech. Rep. LBNL-42953, Lawrence Berkeley National Laboratory (1999).
[14] K. Wu, H. Simon, Thick-restart Lanczos method for large symmetric eigenvalue problems., SIAM J. Matrix Anal. Appl. 22 (2000) 602–616.
[15] Z. Bai, L. Nguyen, Towards scalable computation of many eigenpairs of symmetric matrices, presented at SIAM Conference on Parallel Processing for Scientific Computing (2018).
[16] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM J. Sci. Comput. 23 (2002) 2165–2182.
[17] Z. Bai, D. Hu, L. Reichel, A Newton basis GMRES implementation, IMA J. Numer. Anal. 14 (1994) 563–581.
[18] M. Mohiyuddin, M. Hoemmen, J. Demmel, K. Yelick, Minimizing communication in sparse matrix solvers, in: Proc. of the Conf. on High Perf. Comput. Networking, Storage and Analysis (SC), 2009, pp. 36:1–36:12.
[19] I. Yamazaki, M. Hoemmen, P. Luszczek, J. Dongarra, Improving performance of GMRES by reducing communication and pipelining global collectives, in: IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1118–1127.
[20] C. Paige, Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix, IMA J. Appl. Math. 18 (3) (1976) 341–349.
[21] E. Carson, J. W. Demmel, Accuracy of the s-step Lanczos method for the symmetric eigenproblem in finite precision, SIAM J. Matrix Anal. Appl. 36 (2) (2015) 793–819.
[22] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, SIAM J. Sci. Comput. 34 (2012) A206–A239.