SAND2019-5616C

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

Zhaojun Bai (UCD), Jack Dongarra (UTK),
Ding Lu (UCD), and Ichitaro Yamazaki (SNL)

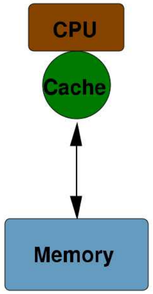
presented by Hartwig Anzt (UTK)

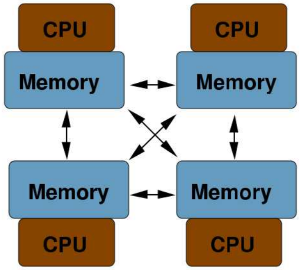
IEEE International Parallel and Distributed Processing Symposium (IPDPS),
Rio de Janeiro, Brazil, May 15, 2019

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Lanczos for large-scale Hermitian Eigenvalue problems

- Lanczos is a powerful method for solving large SYEV, Av = λv
  - used is many applications
    - effective for computing a few exterior eigenvalues (and eigenvectors)
      - several approaches to improve convergence e.g., thick-restart
    - applicable for interior eigenvalues with spectrum transformation (e.g., shift-invert)
  - use two main kernels (based on Krylov subspace projection)
    - Matrix Vector multiply (SpMV)
      for generating Krylov subspace = span(q, Aq, A^2q, ...)
      - often, black box, provided by users
    - Orthgonalization
      for generating orthonormal basis vectors
      - our current focus

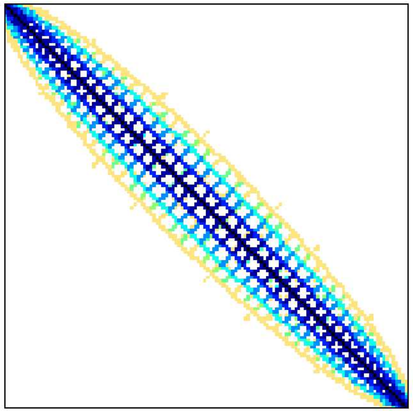




  - Communication can be expensive (time, and maybe power)
    - P2P + irregular data access for SpMV
    - all-reduce + BLAS-1 or 2 for Orthogonalization
    - becoming more expensive on a newer architecture

Challenges in computing many eigenvalues

- Some applications require many eigenvalues (e.g., >1% of n)
  - electronic structure calculation, normal-mode analysis in structure analysis, etc.



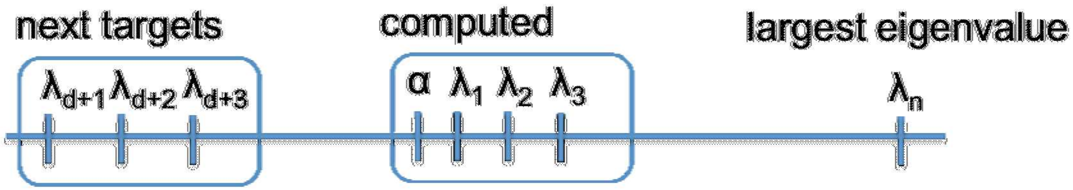
- Other approaches exist
  - Full eigenvalue decomposition
    - ScaLAPACK, ELPA, EigenExa, etc.
    - Stable, but expensive $O(n^3)$
  - Spectral Slicing
    - SLEPc, EVSL, z-Parse, FEAST, etc.
    - Scalable, but several parameters (e.g., windows) and duplicate/missing eigenvalues on the interface
- Lanczos: it is a challenge both numerically & computationally
  - often needs large subspace (e.g., $m=2n_d$)
  - require *locking* (i.e., multiple orthogonalization) to avoid computing the same eigenvalues (nm^2 flops)
  - This talk: combine s-step with EED+TRLan

Explicit external deflation

- Shift the computed eigenvalues away from the exterior:

$$A_d := A + \alpha U_d U_d^T$$

  where U_d contain computed eigenvectors



- Two issues
  - Numerical stability / accuracy (e.g., effects of the errors in the computed eigenvalues on the accuracy of the next eigenvalues to be computed)

    ➔ on-going studies

  - Performance of matrix powers kernel with sparse-plus-low-rank matrix

$$(A + \alpha U_d U_d^T)^k p_0 \qquad \text{for } k = 1, 2, \ldots, s$$

    - deflation of U_d becomes expensive as more eigenvectors are computed

      ➔ focus of this talk (SpMV as a black-box)

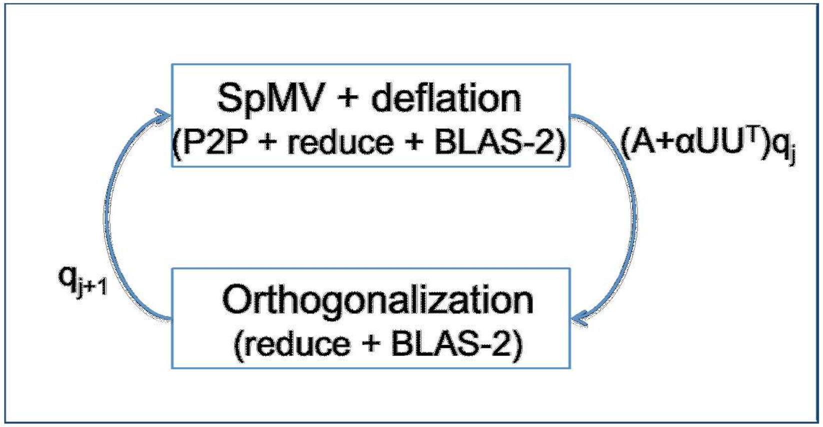
Matrix powers kernel for sparse plus low-rank matrix

- standard kernel

  for j=1, 2, …, s

  - $p_j := A\, p_{j-1} + \alpha U_d\, (U_d^T\, p_{j-1})$

  end



- Each step needs communication
  - p2p for SpMV, and global-reduce for deflation
  - BLAS-2 kernels (SpMV or GEMV)
- "Communication-avoiding" (CA) kernel
  - One global-reduce per every s steps
  - Potential to reduce the communication latency by a factor of s

Matrix powers kernel for sparse plus low-rank matrix

- specialized CA kernel
  - if the computed eigenpairs satisfy (exact and orthogonal)

    $AU_d = U_d \Lambda_d$ and $U_d^T U_d = I$

  - then recurrence for deflation can be un-rolled

$$
\begin{aligned}
p_j &:= (A+\alpha U_d U_d^T)^j p_0 \\
&:= A\,(A+\alpha U_d U_d^T)^{j-1} p_0 + \alpha U_d U_d^T\,(A+\alpha U_d U_d^T)^{j-1} p_0 \\
&:= A\,(A+\alpha U_d U_d^T)^{j-1} p_0 + \alpha U_d(\Lambda_d + \alpha\, I)\, U_d^T (A+\alpha U_d U_d^T)^{j-2} p_0 \\
&:= A p_{j-1} + \alpha U_d(\Lambda_d + \alpha I)^{j-1} U_d^T p_0
\end{aligned}
$$

    - SpMV with the previous vector p_{j-1}
    - GEMV with the starting vector p_0, followed by small local computation

      ➔ One all-reduce per s steps

Matrix powers kernel for sparse plus low-rank matrix

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

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

- Specialized kernel
  1. only one dot-product through GEMV
     - not GEMM
  2. small local computation with small $W_j=(\Lambda+\alpha I)^j$ for deflation
  3. GEMM with local vectors
  4. followed by matrix-powers kernel with sparse matrix A

less communication (sx) and computation (2x) for deflation

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

Accuracy of computed eigenvalues

- Computed eigenpairs are not exact
  - $U_d^T U_d = I + F$
    where F is the orthogonalization error
  - $A\, U_d = U_d\, \Lambda_d + E$
    where E is determined by the stopping criteria

- If the orthogonality is maintained (e.g., two classical Gram Schmidt), then norm of F is small

- It can be shown that if the computed eigenvalues satisfy

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

then the effects of the errors in the computed eigepairs is small in the evaluation of MPK (i.e., the same order as the round-off errors)

General CA MPK: blocking cover, N. Knight, E. Carson, J. Demmel 2014

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

$B := U_d^H \cdot [\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{s-1}]$

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

$Y := U_d \cdot [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_s]$

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
```

- Matrix-powers kernel for a general sparse-plus-low-rank matrix, $A+\alpha U_d U_d^T$
- No assumption on properties of A or U
- Require additional costs, s-1 additional SpMV and then block dot-product with GEMM
  - specialized CA MPK does not require additional SpMV, and perform GEMV instead of GEMM
- Can be applied for TRLan+EED
  (first performance studies of blocking cover for practical application)

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

Putting all together: s-step TRLan+EED

Krylov subspace generation

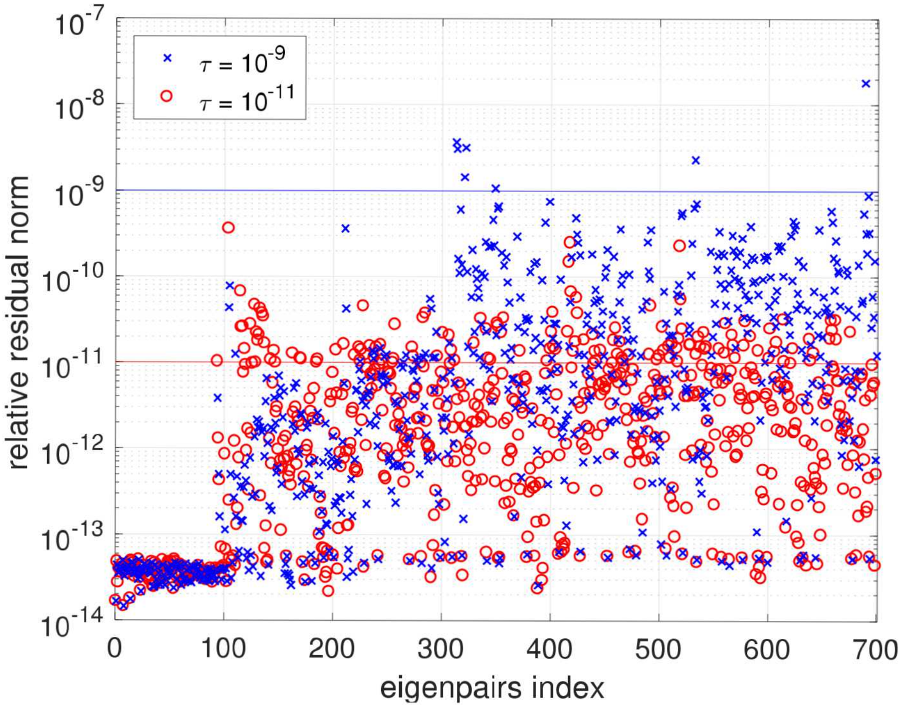
- Matrix powers kernel with sparse-plus-low-rank matrix
  - Standard kernel
  - Specialized/blocking cover CA kernel
- Block orthogonalization
  - block Gram Schmidt: orthogonalize s vectors against previous vectors at once (single all-reduce)
  - Cholesky QR: orthogonalize s vectors among themselves (single all-reduce)

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:
      $B := P_{i+1:i+s}^H P_{i+1:i+s}$
      $R_{i+1:i+s,\, i+1:i+s} := \mathrm{chol}(B)$
      $Q_{i+1:i+s} := P_{i+1:i+s} R_{i+1:i+s,i+1:i+s}^{-1}$
    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

Experimental setups

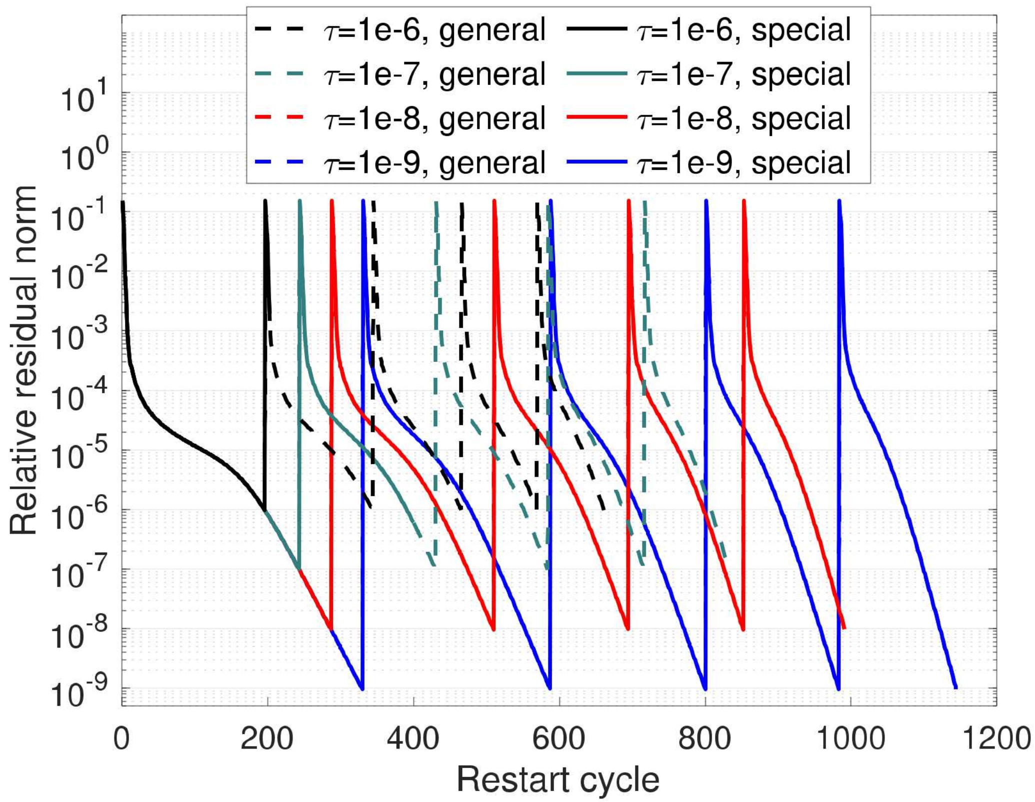
- NERSC Cori Haswell nodes
  - Each node with Intel Xion E5 at 2.3GHz with 128GB of main memory
  - Nodes are connected through Cray Aries (Dragonfly)
- Compiled using the Cray compiler wrapper
  - Linked to Intel's MKL
- Solver parameters
  - Shift is chosen based on the next target and the largest computed eigenvalue, $\alpha=\lambda_d+(\lambda_n-\lambda_d)/2$
  - Eigenpairs are considered to be converged with $\tau=10^{-11}$
  - One MPI process per core

Accuracy of computed eigenpairs with standard TRLan+EED



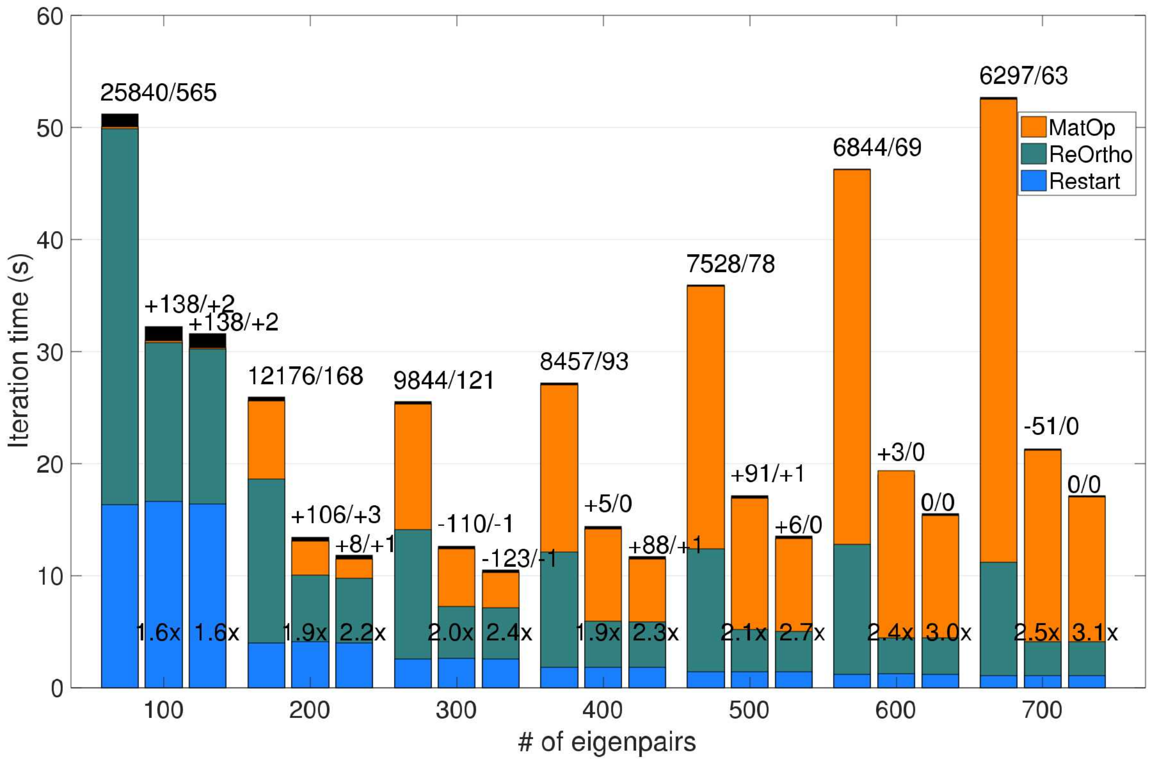
- Relative residual norms with TRLan+EED
  - Compute 100 eigenpairs at a time
  - SiH_4 for DFT electronic structure calculation from Suite Sparse matrix collection
  - TRLan+EED obtain desired accuracy

Accuracy of computed eigenvalues



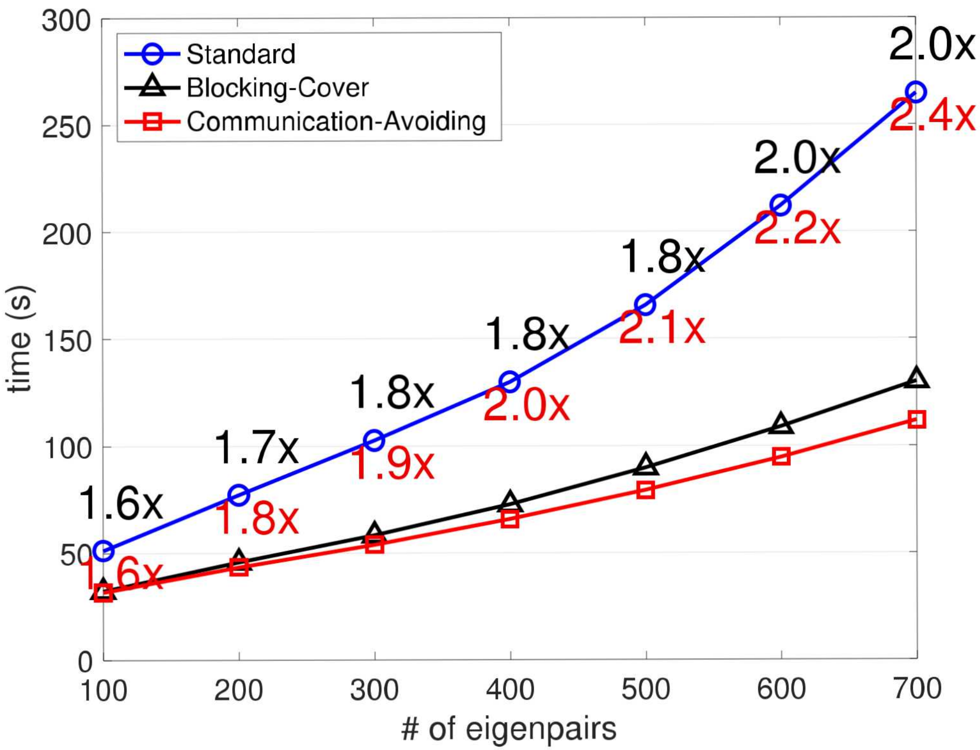
- Specialized MPK is as accurate as standard MPK when tolerance is selected carefully

Performance results using a diag($1^2,2^2,\ldots,n^2$) with n=10K



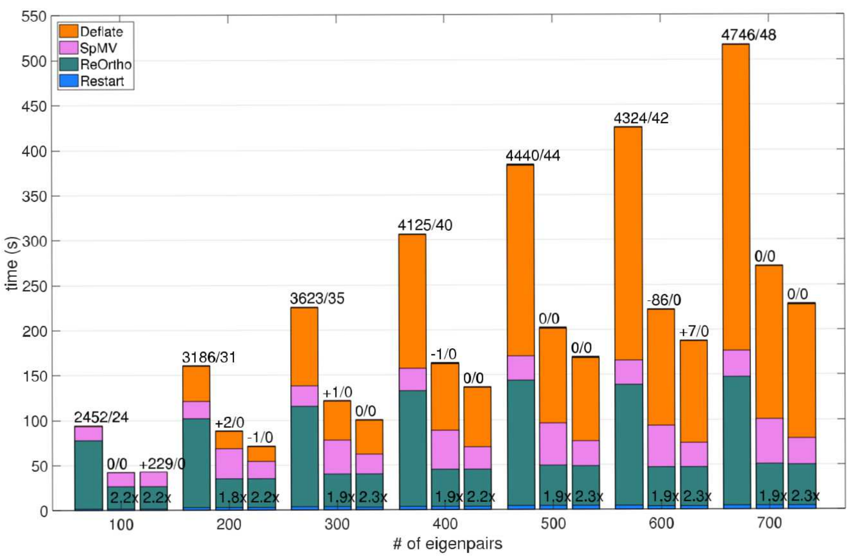
- ReOrtho time reduced through block orthogonalization
  - For computing first 100 eigenpairs, TRLan vs Ca-TRLan
- MatOp time reduced through MPK

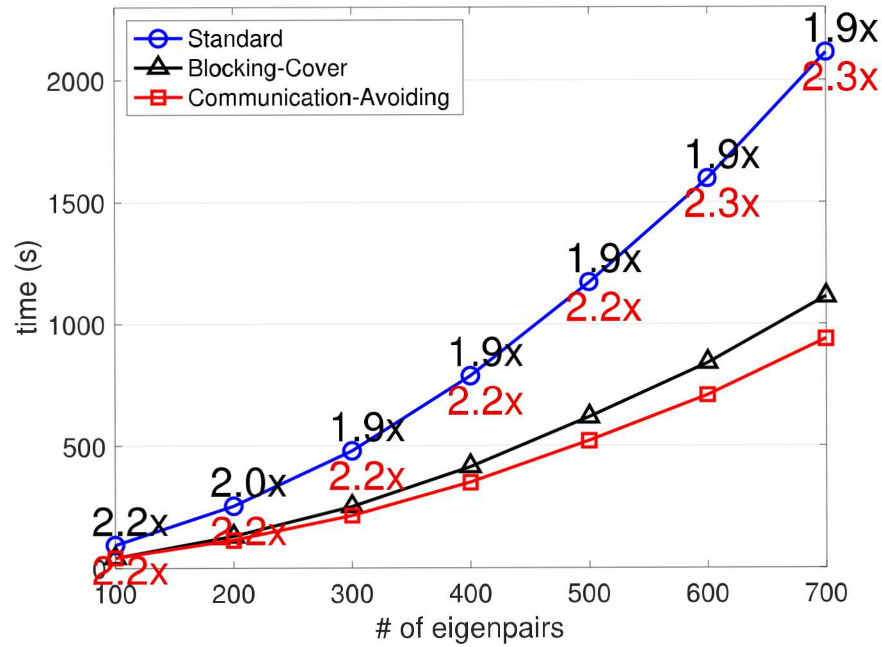
Performance results using a diag($1^2, 2^2, \ldots, n^2$) with n=10K



- Increasing benefits as more eigenvalues are needed

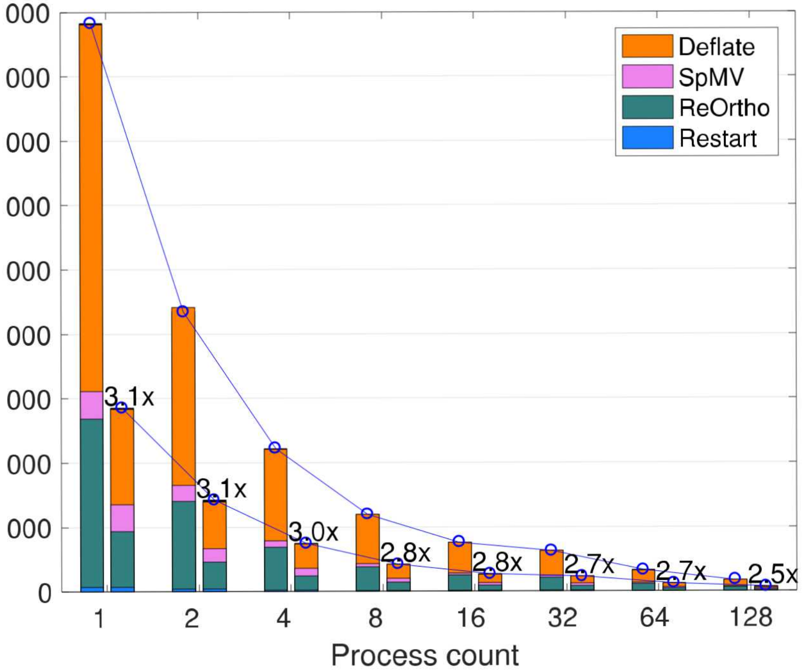
Performance results using a DFC matrix (n=97K)

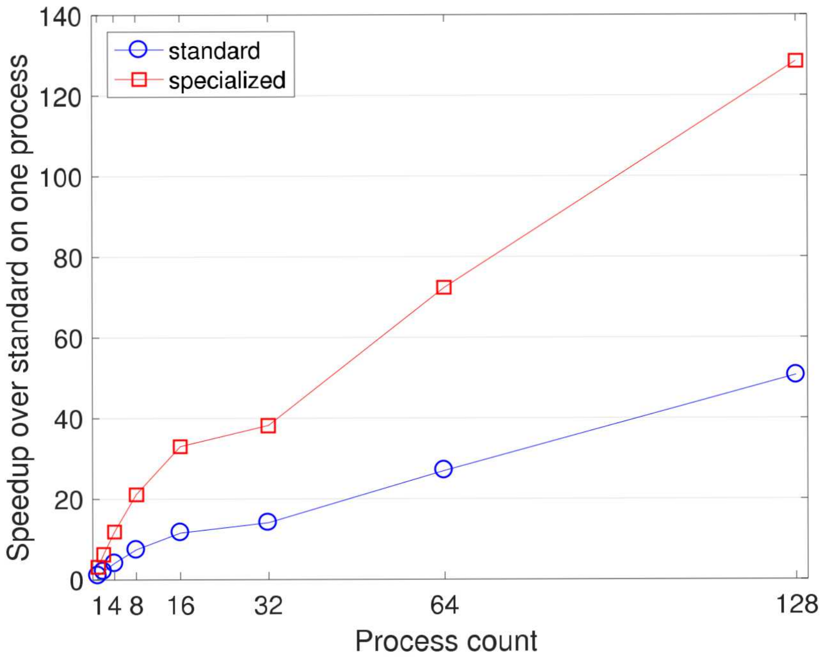




- Avoiding communication/computation can reduce the run time
  - Speedups of up to 2.3x

Performance results using a DFC matrix (n=240K)





- CA variants can maintain the performance benefits over multiple processes
  - Needs to address sequential part (e.g., restart)

Conclusion

- TRLan+EED for computing many eigenvalues
- s-step method for improving performance by avoiding communication
- Possible to avoid some computation when the tolerance is carefully selected

- More theoretical, numerical, and performance studies are underway
  - Low-synchronous orthogonalization kernels
  - Effects of inaccurate eigenpairs on EED

Thank you!!

- ECP PEEKS, for funding

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nations exascale computing imperative.