Performance of Several Lanczos Eigensolvers with HISQ Fermions

Hwancheol Jeong[1,†], Steven Gottlieb[1], Carleton DeTar[2]

[1] *Indiana University,* [2] *University of Utah*

Abstract

We investigate several Lanczos eigensolvers in the Grid and QUDA libraries. We use them to calculate eigenvalues and eigenvectors of the HISQ Dirac operator and analyze their performance.

Simulation environment

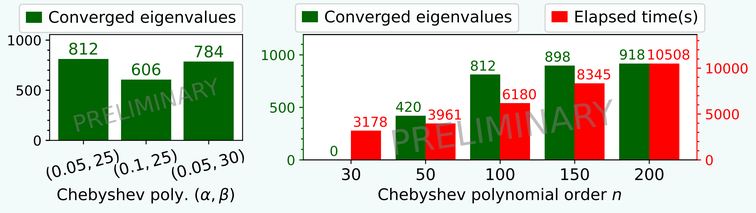
- N_f=2+1+1 MILC HISQ lattice: $24^3\times64$, $a\simeq0.12$ fm, am_l/am_s=0.00507/0.0507
- Valence quark: **HISQ staggered quarks**
- CPU(Grid): Xeon E5-2650v2 (**Octa-core**, **AVX**)
- GPU(QUDA): NVIDIA Tesla **K80** (**Dual**-GPU)
- Tolerance of residual of eigenvalue equation: 10^{-12}.

Lanczos iteration

- **Lanczos algorithm** [1]: **Hermitian** H → **tridiagonal** T
  - $\mathcal{K}_n(H,b) = \mathrm{span}\{b, Hb, H^2b, \cdots, H^{n-1}b\}$: Krylov subspace
  - $\{q_i\text{'s}\}$: basis of $\mathcal{K}_n(H,b)$, generated by Gram-Schmidt process
  - $T_{n+1,n}\,q_{n+1} = Hq_n - T_{n,n}\,q_n - T_{n-1,n}\,q_{n-1}$
  - For $Q_n = (q_1|q_2|\cdots|q_n)$, $T_n = {Q_n}^\dagger H Q_n$
- Submatrix T_n has n eigenvalues approximate to *some* n eigenvalues of H. (some: largest, smallest, least dense)
- As n increases, **eigenvalues of T_n converge to true eigenvalues of H.** ⇒ **Lanczos iteration method**
- Diagonalization of T_n: QR iteration method ⇒ $T_n = V_n \Lambda {V_n}^\dagger$
- Eigenvector estimates (Ritz vector): $W_n = Q_n V_n$ where $W_n = (w_1|w_2|\cdots|w_n)$
- Eigenvalue estimates (Ritz value): $\Lambda_{i,i}$ or $\tilde{\Lambda}_{i,i} \equiv \dfrac{\langle w_i|H|w_i\rangle}{\langle w_i|w_i\rangle}$
- Residual: $\dfrac{\left\| H|w_i\rangle - \tilde{\Lambda}_{i,i}|w_i\rangle \right\|}{\sqrt{\langle w_i|w_i\rangle}}$ (or divide by some normalization factor)

Polynomial acceleration

- Convergence speed of Lanczos iteration depends on eigenvalues' density. ⇒ **less dense, converge faster**
- **Polynomial acceleration** [2]: manipulate eigenvalue density by applying a polynomial. (does not alter eigenvectors)
- **Chebyshev polynomial** $C_n(x)$
  - n: polynomial order
  - $\boldsymbol{\alpha \le x \le \beta}$: $\sim \cos(nx)$, bounded, **dense** ⇐ **unwanted** eigenvalues
  - $\boldsymbol{x < \alpha}$ or $\boldsymbol{x > \beta}$: $\sim \cosh(nx)$, diverge, **sparse** ⇐ **wanted** eigenvalues
- Eigenvalue convergence with Chebyshev polynomial by 1000 Lanczos iterations ((largest eigenvalue) $\lesssim 25$, (1000-th eigenvalue) $\lesssim 0.05$)

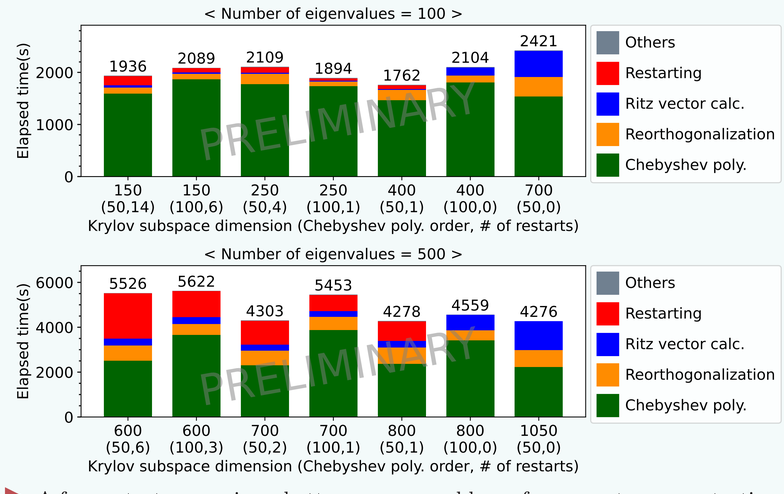


- Optimization of Chebyshev parameters are under investigation.

Implicitly restarted Lanczos

- $\mathcal{K}_n(\,H, (H-\mu)b\,)$ does not contain an eigenvector for eigenvalue μ.
- **Implicit restart** [3]: exclude unwanted eigenvalues
  1. Run n steps of Lanczos on $\mathcal{K}_n(H,b)$.
  2. Calculate n eigenvalue estimates. Choose k unwanted eigenvalues μ_i's.
  3. Rotate basis by which Lanczos restarts from $(n-k)$-th step as if the starting vector was $(H-\mu_k)\cdots(H-\mu_1)b$.

  ⇒ **Exclude unwanted eigenvalues** μ_i's.
  ⇒ Improve remaining (wanted) eigenvalues' convergence.
- Grid's `ImplicitlyRestartedLanczos` [4, 5] (MPI: 1, OpenMP: 8)

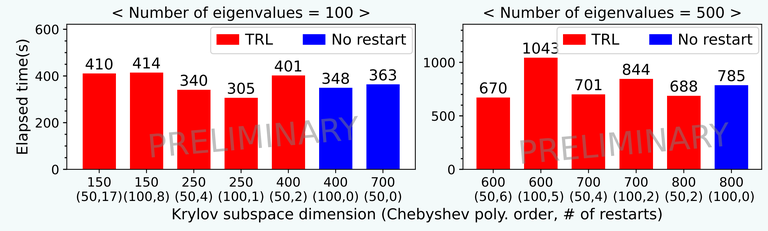


- A few restarts may give a better or comparable performance to non-restarting.
- Optimizing Chebyshev polynomial and Krylov subspace are essential.

Thick restarted Lanczos

- **Thick restart** [6]: keep wanted eigenvalues
  1. Run n steps of Lanczos on $\mathcal{K}_n(H,b)$.
  2. Calculate n eigenvalue estimates. Find k (nearly) converged eigenvalues.
  3. Rotate basis to be k Ritz vectors.
  4. Restart constructing basis of $\mathcal{K}_{n-k}(H, q_{n+1})$ orthogonal to k Ritz vectors.

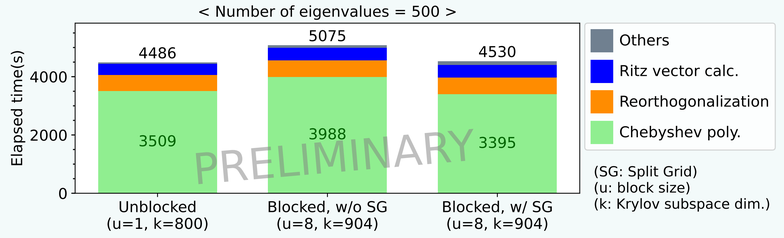
  ⇒ **Keep converged eigenvectors**. Restarted Lanczos searches on subspace orthogonal to them.
  ⇒ Improve remaining eigenvalues' convergence.
- In practice, we **keep a thick (converged + more) number of eigenvalues**.
- Implement **locking**: deflate eigenvectors converged to machine precision
- QUDA's `TRLM` [7, 8] (MPI(GPU): 2)



- A proper number of restarting may improve the performance. ⇒ Need tuning.
- Using less memory is also beneficial for GPUs.

Block Lanczos with Split Grid

- **Block Lanczos**: Lanczos with multiple starting vectors $b_{i=1,\cdots,u}$.
  - Hermitian H → block-tridiagonal T
  - $\mathcal{K}_n(H,b_1)\cup\cdots\cup\mathcal{K}_n(H,b_u)$ $= \mathrm{span}\{b_1,\cdots,b_u, Hb_1,\cdots,Hb_u,\cdots,H^{n-1}b_1,\cdots,H^{n-1}b_u\}$.
  - Each iteration computes a set of basis from $H^ib_1,\cdots,H^ib_u$.
- **Split Grid**: Split jobs into multiple smaller MPI grids so that each split grid has smaller surface to volume ratio. ⇒ less communication per computation
- **Block Lanczos with Split Grid** [9] : split operations of H to $H^{i-1}b_1,\cdots,H^{i-1}b_u$ (or previous basis set).
- Grid's `ImplicitlyRestartedBlockLanczos` [4, 10] (MPI: 8 (**intra-node**), OpenMP: 1)



- Block Lanczos converges slower, but Split Grid makes Dirac operation faster.
- May perform better on **inter-node** networks. (We plan to test!)
- QUDA's `BLKTRLM` [7, 8]: In development. Wait for MRHS code implementation.

Conclusion

- Chebyshev polynomial improves Lanczos iteration's convergence. It is essential to tune its parameters considering computational performance and convergence.
- Chebyshev polynomial's order and Krylov subspace's dimension need to be tuned to optimize restarted Lanczos algorithms' performance.
- With an optimized Chebyshev polynomial, restarted Lanczos algorithms may not perform significantly better than non-restarted Lanczos algorithm. However, it still allows using less memory.
- Performance of the block Lanczos utilizing the Split Grid method is similar to that of the unblocked Lanczos. (Need check on inter-node network.)
- MILC code[11] provides interfaces for Grid's `ImplicitlyRestartedLanczos` and QUDA's `TRLM`.

References

[1] Cornelius Lanczos. *J. Res. Natl. Bur. Stand. B Math. Sci.*, 45:255–282, 1950.
[2] Youcef Saad. *Math. Comp.*, 42(166):567–588, 1984.
[3] Danny C. Sorensen. volume 4 of *ICASE/LaRC Interdiscip. Ser. Sci. Eng.*, pages 119–165. Kluwer Acad. Publ., Dordrecht, 1997.
[4] Peter A. Boyle et al. *PoS*, LATTICE2015:023, 2016.
[5] `https://github.com/paboyle/Grid`.
[6] K Wu and H Simon. *SIAM J. Matrix Anal. Appl.*, 22(2):602–616, 2000.
[7] M. A. Clark et al. *Comput. Phys. Commun.*, 181:1517–1528, 2010.
[8] `https://github.com/lattice/quda`.
[9] Yong-Chull Jang and Chulwoo Jung. *PoS*, LATTICE2018:309, 2019.
[10] `https://github.com/paboyle/Grid/tree/feature/block_lanczos`.
[11] `https://github.com/milc-qcd/milc_qcd/tree/develop`.

† speaker, sonchac@gmail.com