Performance of several Lanczos eigensolvers with HISQ fermions

Hwancheol Jeong,[a,*] **Carleton DeTar**[b] **and Steven Gottlieb**[a]

[a] *Department of Physics, Indiana University, Bloomington, Indiana 47405, USA*

[b] *Department of Physics and Astronomy, University of Utah, Salt Lake City, Utah 84112, USA*

E-mail: sonchac@gmail.com, detar@physics.utah.edu, sg@indiana.edu

We investigate the state-of-the-art Lanczos eigensolvers available in the Grid and QUDA libraries. They include Implicitly Restarted Lanczos, Thick-Restart Lanczos, and Block Lanczos. We measure and analyze their performance for the Highly Improved Staggered Quark (HISQ) Dirac operator. We also discuss optimization of Chebyshev acceleration.

The 38th International Symposium on Lattice Field Theory, LATTICE2021 26th-30th July, 2021 Zoom/Gather@Massachusetts Institute of Technology

*Speaker

1. Introduction

Recent improvements in lattice QCD calculations have been achieved by making use of the eigenvalue spectrum of lattice operators, such as the lattice Dirac operator and the lattice Laplacian operator. The Lanczos iteration method is an eigensolver algorithm employed for that purpose [1]. It enables calculation of both eigenvalues and eigenvectors of a Hermitian matrix. It is most useful when a partial eigenvalue spectrum is required. Additional iterations provide more eigenmodes with higher accuracy.

There are several improvement methods for the Lanczos algorithm: (1) Polynomial acceleration applies a polynomial to the matrix of interest so that the Lanczos iteration converges faster with a transformed eigenvalue spectrum. Chebyshev polynomials are a popular choice [2]. (2) There are two kinds of restarted Lanczos algorithms that take advantage of information learned from the previous run. The Implicitly Restarted Lanczos (IRL) algorithm suppresses unwanted eigenvalues at the restart [3]. Alternatively, the Thick-Restart Lanczos (TRL) algorithm suppresses or explicitly deflates converged wanted eigenvalues [4]. (3) There are approaches that utilize blocking to improve the computational efficiency of the Lanczos algorithm. The Block Lanczos (BL) algorithm can enhance the compute-to-communication ratio with the Split Grid method [5].

Grid [6, 7] and QUDA [8–10] are very popular and highly performing lattice QCD libraries, especially for modern parallel computing systems. For the Lanczos eigensolver, Grid has IRL and BL eigensolvers. QUDA has a TRL eigensolver and its block variant. All of them utilize Chebyshev polynomials to enhance the convergence.

We investigate these Lanczos eigensolver algorithms in the Grid and the QUDA libraries measuring and analyzing their performance for the Highly Improved Staggered Quark (HISQ) Dirac operator. In Section 2, we describe our simulation details. In Section 3, we describe some basics of the Lanczos iteration method. Section 4 describes the general strategy of utilizing Chebyshev polynomials and studies its optimization. In Sections 5 and 6, we illustrate ideas of IRL and TRL, respectively, and discuss their optimizations. In Section 7, we examine the performance of BL with the Split-Grid method. We conclude in Section 8.

2. Simulation details

We calculate eigenvalues and eigenvectors of the low modes of the massless HISQ Dirac operator D. The Lanczos eigensolvers are run using a Hermitian variation $D^\dagger D$, which is (semi)-positive definite that allows even-odd splitting. The eigensolver performance is measured in elapsed time taken by the eigensolver routine itself. We use an $N_f = 2+1+1$ MILC HISQ gauge configuration, with lattice size $24^3 \times 64$, lattice spacing $a \approx 0.12$ fm, and $am_l/am_s = 0.00507/0.0507$ [11].

We run Grid on Intel CPUs and QUDA on NVIDIA GPUs, where they perform their best so that we can focus on algorithmic performance. All Grid runs are done on a single node of dual Intel Xeon E5-2650v2 system that has $2 \times 8 = 16$ CPU cores, where we use only half, *i.e.,* eight, of them for performance stability. We apply OpenMP for their parallelization, except for the BL eigensolver in Section 7, where we apply MPI. QUDA runs are done on the same system but with two NVIDIA K80 GPUs.

3. Lanczos algorithm and Lanczos iteration method

Let us consider an $m \times m$ Hermitian matrix A, and an m-dimensional vector b. The Gram-Schmidt process on a set of vectors $\{b, Ab, A^2 b, \cdots, A^{n-1} b\}$ for $n \leq m$, which spans a Krylov subspace $\mathcal{K}_n(A, b)$, gives a three term recurrence relation

$$t_{i+1,i}\, q_{i+1} = A q_i - t_{i,i}\, q_i - t_{i-1,i}\, q_{i-1}\,, \tag{1}$$

where q_i are orthonormal basis vectors of $\mathcal{K}_n(A, b)$, and $t_{ij} = \langle q_i \,|\, A q_j \rangle$. Combining recurrences of $i = 1, \cdots, n$, we have

$$T_n = Q_n^\dagger A Q_n\,, \tag{2}$$

where $Q_n = (q_1 | q_2 | \cdots | q_n)$, and $T_n = (t_{ij})$ is an $n \times n$ Hermitian tridiagonal matrix.

The Lanczos algorithm is to construct T_n and Q_n by iterating the Lanczos recurrence relation Eq. (1) [1]. When $n = m$, Q_m is a unitary transformation, so T_m has the same eigenvalue spectrum as A. Even for $n < m$, eigenvalues of T_n are approximate to some n eigenvalues of A [12]. They converge to true eigenvalues of A as n increases. The largest, the smallest, or the most isolated (the least dense) extreme eigenvalues appear first and converge first. When $k(< m)$ eigenvalues are wanted, we need to perform n ($k \leq n \leq m$) iterations of the Lanczos recurrence (Eq. (1)) with the result that T_n has eigenvalues approximate to k eigenvalues of A of interest within the target precision.

A QR iteration is usually employed to diagonalize a tridiagonal matrix T_n [13, 14]. Then its eigenvalue decomposition gives $T_n = V_n \Lambda V_n^\dagger$, where Λ is a diagonal matrix with eigenvalues λ_i of T_n, and V_n is composed of their corresponding eigenvectors v_i. With this, we calculate eigenvector estimates (called Ritz vectors) w_i and eigenvalue estimates (called Ritz values) $\tilde{\lambda}_i$ by

$$w_i = Q_n v_i\,, \quad \tilde{\lambda}_i = \frac{\langle w_i | A | w_i \rangle}{\langle w_i | w_i \rangle}\,. \tag{3}$$

The convergence of each eigenvector is measured by a residual

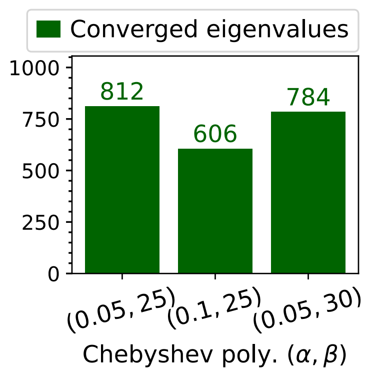
$$\frac{\left\| A | w_i \rangle - \tilde{\lambda}_i | w_i \rangle \right\|}{\sqrt{\langle w_i | w_i \rangle}}\,, \tag{4}$$

or dividing it by some normalization factors. Note that $\tilde{\lambda}_i$ can be different from λ_i since we usually apply a polynomial to A, as will be discussed in Section 4.

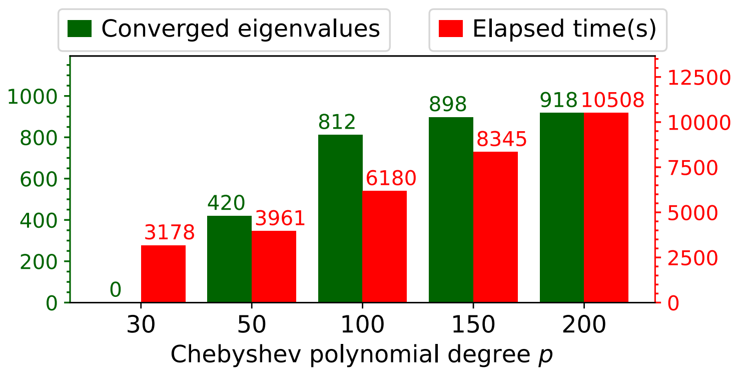
4. Chebyshev acceleration

The convergence of a Lanczos iteration depends on the eigenvalue density. Less dense (including the largest and the smallest) eigenvalues appear early and converge fast. One can control the density of eigenvalues, and, thus, the convergence of Lanczos iteration, by applying a polynomial $\mathcal{P}$ to the matrix A. It maps eigenvalues λ_i of A into $\mathcal{P}(\lambda_i)$ but preserves their eigenvectors.

Chebyshev polynomials $C_p(x)$ of degree p are bounded within $[-1, 1]$ for $|x| \leq 1$, while diverging rapidly for a high p as $|x|$ increases for $|x| > 1$ [15]. We can transform it to map unwanted eigenvalues into the bounded region and wanted eigenvalues into the diverging region.



(a)



(b)

Figure 1: The convergence of Chebyshev accelerated Lanczos iterations for different Chebyshev parameters (α, β) (left) and degree p (right).

This Chebyshev acceleration can boost the convergence of Lanczos iteration. With suitable choices of the location of the bounded region and the polynomial degree, its benefit more than compensates for the cost of applying the polynomial.

Eigenvalues of $D^\dagger D$ are non-negative, and our usual interests lie in its low modes. We transform the domain of $C_p(x)$ so that $[-1, 1]$ becomes $[\alpha, \beta]$ that covers the unwanted eigenvalues. The value of α should be set to be greater than the largest desired eigenvalue, and β to be greater than the largest eigenvalue in the whole spectrum. A few power iterations can help to estimate the size of the largest eigenvalue. Choosing α needs some heuristics, but we may use the same value within a gauge ensemble.

In Fig. 1, we show the convergence of Lanczos iterations with different Chebyshev acceleration parameters. Here, Grid's `ImplictlyRestartedLanczos` eigensolver is used with a little modification not to restart. We run 1000 Lanczos iterations, and the convergences are checked by Eq. (4) in double precision. Figure 1a presents the number of converged eigenvalues for different α and β values. The same Chebyshev polynomial degree $p = 100$ is used here, which means all the runs take the same time in principle. On this gauge configuration, the largest eigenvalue of $D^\dagger D$ is around 21, and the 1000th eigenvalue is around 0.046. The result indicates that we get the best convergence when both α and β are the closest to the 1000th eigenvalue and the largest eigenvalue, respectively. The 800th eigenvalue is around 0.034. Hence, $\alpha = 0.04$ might give a better convergence on this gauge configuration. However, a tight choice may not work for other gauge configurations in the same ensemble.

In Fig. 1b, we present the results for different polynomial degrees p. Here, we used $(\alpha, \beta) = (0.05, 25)$, the best pair found in Fig. 1a. Although higher p makes more eigenvalues converge, it demands more computations. In this example, $p = 100$ gives the best efficiency. In general, the most efficient p depends on the number of eigenvalues we want. We need heuristics to find an optimized p.

Every improved Lanczos algorithm discussed in this paper utilizes Chebyshev acceleration. We use $(\alpha, \beta) = (0.05, 25)$, and $p = 50$ or 100.

5. Implicitly Restarted Lanczos

A basic Lanczos algorithm must carry on its iteration by increasing the Krylov subspace dimension n until eigenvalues of the desired number converge. As n increases, it takes more memory to store the eigenvectors as well as more computing time for the iteration[1] and for the diagonalization of a larger T_n. Moreover, it is difficult to anticipate an optimized (or the smallest) n without monitoring the convergence, which includes diagonalizing T_i ($i < n$) and computing eigenvectors and their residuals.

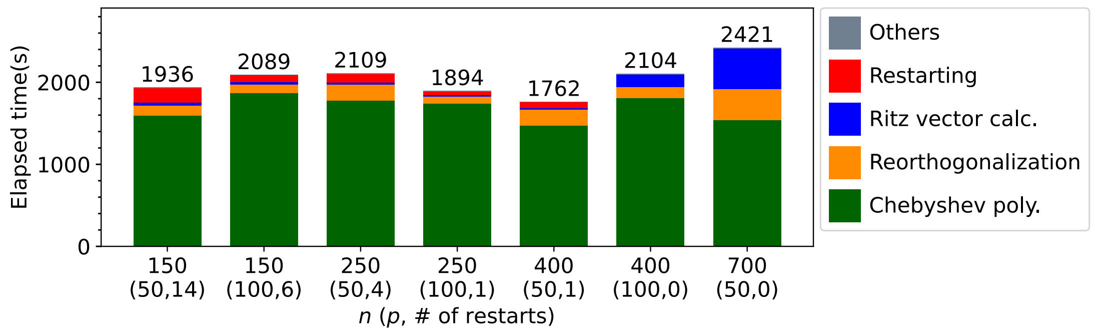
Suppose we have performed n Lanczos iterations for Hermitian matrix $A(= C_p(D^\dagger D))$ with a starting vector b. The Implicitly Restarted Lanczos (IRL) algorithm allows us to restart from the $(n-k)$-th step of the Lanczos algorithm for A with a new starting vector $\tilde{b} = (A-\mu_k)\cdots(A-\mu_1)b$, where the $\{\mu_i, i = 1, \dots k\}$ are some eigenvalues [3]. The Krylov subspace $\mathcal{K}_n(A, \tilde{b})$ for the restarted run does not contain eigenvectors corresponding to the eigenvalues μ_i. It not only excludes unwanted eigenvalues and eigenvalues near them, but it also improves the convergence of the remaining wanted eigenvalues far from the $\{\mu_i\}$.

When we are interested in only the t lowest eigenmodes, a typical implementation of IRL runs the Lanczos iteration to the n-th step, does the implicit restart that excludes the k largest eigenvalues found in the previous run, and repeats them until the t smallest eigenvalues converge. In this way, we can constrain the Krylov subspace dimension to any $n > t$, as much as the system memory allows. However, since k is usually set to make $n-k$ slightly larger than t, n determines how many restarts we need to do. Each restart must perform the QR decomposition k times and do the basis rotations using them. This cost can be comparable to that of many Lanczos iterations. Hence, we need to find an optimized n that maximizes the restarting efficiency.

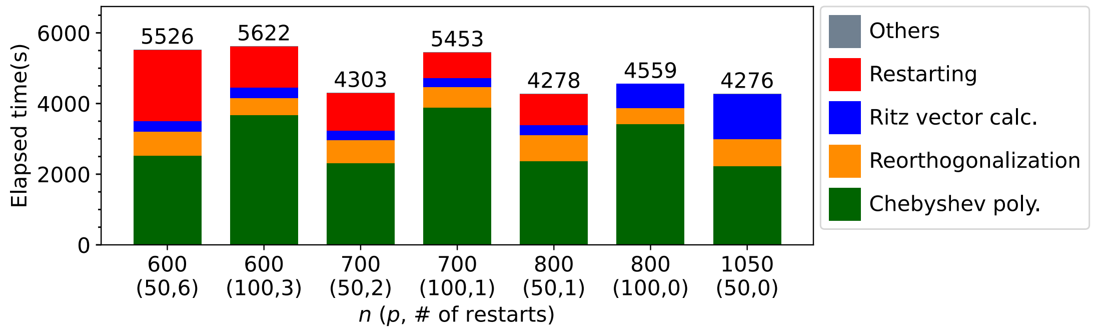
The Grid library has an IRL eigensolver named `ImplicitlyRestartedLanczos`. In Fig. 2, we present its performance profiles for various Krylov subspace dimensions n with Chebyshev acceleration of $p = 50, 100$. For non-restarted cases, a modified version mentioned in Section 4 is used. Note that in the Lanczos iteration, eigenvalues converge simultaneously, not one by one in order. Hence, it could be less efficient for calculating a small number of eigenvalues. In Fig. 2a, where we calculate 100 eigenvalues, the non-restarted Lanczos converges at $n = 400$ and $n = 700$ for $p = 50$ and $p = 100$, respectively. In a (relatively) slowly converging case like this, IRL can perform better than the basic non-restarted Lanczos. For both $p = 50$ and 100, we obtain the best performance by restarting once.

On the other hand, in Fig. 2b, where we calculate 500 eigenvalues, the non-restarted Lanczos takes the least time. Here, $n = 800, 1050$ are only about twice 500, so the restarting costs for $n = 600, 700, 800$ are comparable to or more expensive than running Lanczos on a bigger Krylov subspace. However, the differences between $n = 700, 800, 1050$ are not huge for $p = 50$, so one may choose $n = 700$ to save memory. In addition, unlike the non-restarted Lanczos where the convergence is not guaranteed for a given n, IRL assures its convergence when the same n is used for other gauge configurations, even though it may need more restarts.

[1]In practice, Lanczos vectors q_i's orthogonality gets inexact as iteration continues due to the accumulation of numerical errors. One can correct it by re-orthogonalizing all q_i's per some period of iterations. This is expensive for a large n.



(a) (Number of eigenvalues) = 100



(b) (Number of eigenvalues) = 500

Figure 2: Performance of the Grid IRL eigensolver for various Krylov subspace dimensions n with Chebyshev polynomial degrees $p = 50, 100$.

One last remark is that we should optimize the Chebyshev acceleration parameters together with n. In Fig. 2a, the non-restarted Lanczos converges faster with $p = 100$ than $p = 50$, while the IRL's best performance comes with $p = 50$.

6. Thick-Restart Lanczos

In this section, we investigate another restarted Lanczos algorithm called Thick-Restart Lanczos (TRL) [4]. Unlike IRL, which suppresses unwanted eigenvectors to improve convergence, TRL suppresses (nearly) converged wanted eigenvectors from the Krylov subspace for the restarted run.

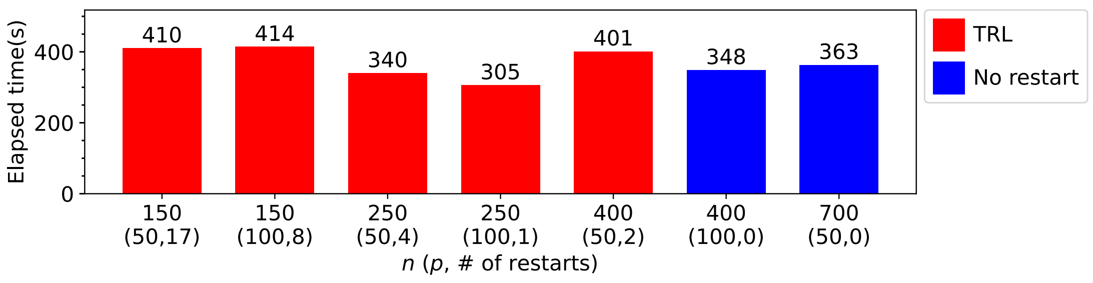
One can rewrite Eq. (2) as

$$\Lambda_k = (Q_n V_k)^\dagger A\, Q_n V_k = W_k{}^\dagger A\, W_k\,, \tag{5}$$

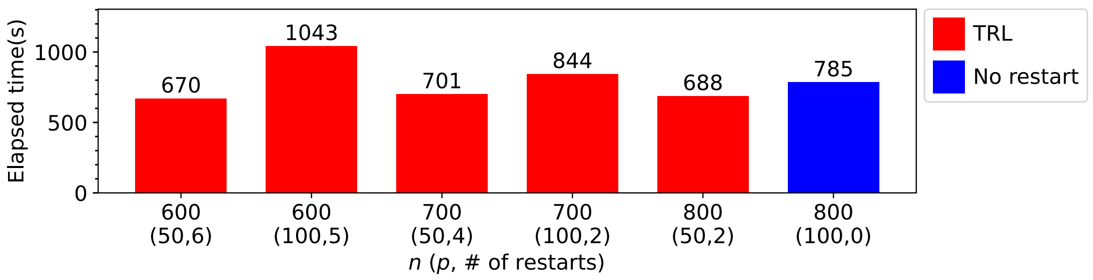
where V_k is composed of some $k (< n)$ eigenvectors of T_n, and $W_k \equiv (w_1|w_2|\cdots|w_k)$ is composed of the corresponding k eigenvector estimates (Ritz vectors) of A. Λ_k is a diagonal matrix of k eigenvalues of T_n that are eigenvalue estimates (Ritz values) of A. Note that w_i's are basis vectors of the same Krylov subspace $\mathcal{K}_n(A, b)$. Hence, we can replace Q_n with W_k, and T_n with Λ_k in Eq. (2).

The TRL algorithm restarts the Lanczos iteration upon W_k and Λ_k, but by appending the vector q_{n+1}.[2] One problem is that w_i's are not orthogonal to q_{n+1}, so $\langle A q_{n+1} | w_i \rangle$ terms survive in the

[2]One cannot start the Lanczos algorithm with an eigenvector, because it generates a rank-1 Krylov subspace.



(a) (Number of eigenvalues) = 100



(b) (Number of eigenvalues) = 500

Figure 3: Performance of the QUDA TRL eigensolver for various Krylov subspace dimensions n with Chebyshev polynomial degrees $p = 50, 100$.

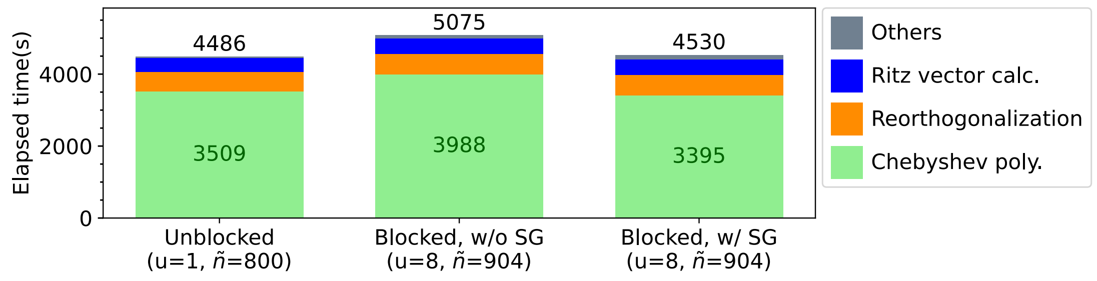
Gram-Schmidt process generating q_{n+2}. This results in a arrow-like matrix T_{n+1}. However, thanks to the symmetry of A, the generation of q_{n+2+i} $(i \geq 1)$ returns to the normal three-term Lanczos recurrence. Hence, we need to deal only with diagonalizing the arrow-like matrix, which can be done in several ways including the general matrix diagonalization.

Now that q_{n+2} is orthogonal to $w_1, \cdots, w_k$, approximate eigenvectors of A, the restarted run searches on the Krylov subspace orthogonal to them. It focuses on finding other eigenvectors rather than w_i's, though it still slowly improves w_i's convergences as well. With a proper choice of k, it can balance the convergence of eigenvalues efficiently.

The QUDA library has a TRL eigensolver named `TRLM`. This eigensolver determines k as the number of converged eigenvalues in the desired precision plus the half of the remaining eigenvalues that are still converging. It also implements the locking method, which explicitly deflates some converged eigenvectors from the restarted run. `TRLM` locks eigenvectors converged to the machine precision. It reduces memory usage and computing cost.

In Fig. 3, we present QUDA `TRLM`'s performance for various Krylov subspace dimensions n with Chebyshev acceleration of $p = 50, 100$. In both Fig. 3a and Fig. 3b, we find TRL can perform better than the non-restarted Lanczos with optimization. One benefit of TRL, compared with IRL, is that its restarting cost can be absorbed into the Ritz vector calculation. That might be why TRL takes the least time for $n = 600$ with six restarts in Fig. 3b.

Restarted Lanczos algorithms (both TRL and IRL) allow us to use less memory. It is especially useful for GPU systems, where memory is usually lacking. Figure 3b does not include the non-restarted Lanczos result for $p = 50$ because it does not converge within the given GPU memory size.



(a) (Number of eigenvalues) = 500

Figure 4: Performance of the Grid BL eigensolver with the Split Grid method (SG), compared with unblocked basic Lanczos and BL without the Split Grid method.

7. Block Lanczos with Split Grid

The Block Lanczos (BL) algorithm runs the Lanczos iteration with multiple (u) starting vectors $\{b_1, b_2, \cdots, b_u\}$, where b_i's are orthogonal to each other [5]. At each j-th iteration of BL, a block version of the Lanczos recurrence Eq. (1) constructs u basis vectors $q^1_j, \cdots, q^u_j$ of the combined Krylov subspace $\mathcal{K}_n(A, b_1) \cup \cdots \cup \mathcal{K}_n(A, b_u)$ by orthogonalizing $Aq^1_{j-1}, \cdots, Aq^u_{j-1}$ simultaneously. After n iterations, it transforms A into a block-tridiagonal matrix T_n of dimension $(nu \times nu)$.

The convergence of a BL iteration is usually slower than that of the basic Lanczos for the same subspace dimension $\tilde{n} = nu$, because the Krylov subspace dimension n per starting vector b_i is smaller than $\tilde{n}$. However, on a parallel computing system, BL may outperform the basic Lanczos by parallelizing the multiplications Aq^i_{j-1} for $i = 1, \cdots, u$. The Split Grid method is one way of doing it [5]. It splits a communication grid such as MPI and distributes parallel jobs into the split grids, so that each job runs with a lower surface-to-volume ratio.

The Grid library has a BL eigensolver implementing the Split Grid method, named `ImplicitlyRestartedBlockLanczos`.[3] In Fig. 4, we compare its performance with the unblocked basic Lanczos and BL without the Split Grid method. The block size for BL is set to $u = 8$. In this measurement, we run 8 MPI processes connected by an intra-node network. We calculate 500 eigenvalues, for which the unblocked Lanczos converges at $\tilde{n} = 800$, while the BL converges at $\tilde{n} = 904$. Hence, without the Split-Grid method, the BL algorithm itself is slower than the basic Lanczos algorithm, as the results show.

However, the Split Grid method can enhance the BL's performance. In Fig. 4, BL's performance is improved by 12% with the Split Grid method, even on the intra-node network. Although it is still a little slower than the unblocked Lanczos, its advantage could become significant on a slow inter-node network.[4]

8. Conclusion

We have discussed improved Lanczos algorithms and their optimizations. A well-tuned Chebyshev polynomial improves the Lanczos iteration's convergence significantly. All other improved

[3]Although it provides both non-restarted and implicitly restarted versions of BL, the latter is not pursued.

[4]It is under investigation. A preliminary result shows that with 8 MPI inter-nodes, BL with Split Grid gives around 200% better performance than the unblocked Lanczos.

Lanczos algorithms utilize it. Restarted Lanczos algorithms, such as Implicitly Restarted Lanczos and Thick-Restart Lanczos, do not always perform better than the non-restarted Lanczos algorithm. Still, we can optimize them to perform better than or comparable to non-restarted Lanczos. It is advantageous for small memory systems such as GPUs. The performance of Block Lanczos utilizing the Split Grid method is comparable to that of the unblocked Lanczos on an intra-node network. It may perform better on inter-node networks.

Acknowledgments

We would like to thank D. Howarth, Y. Jang, and C. Jung for useful discussions. 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. We also gratefully acknowledge support from the Department of Energy grant DE-SC0010120 and from the National Science Foundation grant PHY20-13064. Finally, we thank the developers who support Grid and QUDA whose names can be found at the respective websites for the software.

References

[1] C. Lanczos, *J. Res. Natl. Bur. Stand. B Math. Sci.* **45** (1950) 255.

[2] Y. Saad, *Math. Comp.* **42** (1984) 567.

[3] D.C. Sorensen, vol. 4 of *ICASE/LaRC Interdiscip. Ser. Sci. Eng.*, pp. 119–165, Kluwer Acad. Publ., Dordrecht (1997), DOI.

[4] K. Wu and H. Simon, *SIAM J. Matrix Anal. Appl.* **22** (2000) 602.

[5] Y.-C. Jang and C. Jung, *PoS* **LATTICE2018** (2019) 309.

[6] P.A. Boyle, G. Cossu, A. Yamaguchi and A. Portelli, *PoS* **LATTICE2015** (2016) 023.

[7] `https://github.com/paboyle/Grid`.

[8] M.A. Clark, R. Babich, K. Barros, R.C. Brower and C. Rebbi, *Comput. Phys. Commun.* **181** (2010) 1517 [`0911.3191`].

[9] R. Babich, M.A. Clark, B. Joo, G. Shi, R.C. Brower and S. Gottlieb, 9, 2011, DOI [`1109.2935`].

[10] `https://github.com/lattice/quda`.

[11] MILC collaboration, *Phys. Rev. D* **87** (2013) 054505 [`1212.4768`].

[12] L.N. Trefethen and I. David Bau, *Numerical Liniear Algebra* (1997).

[13] J.H. Wilkinson, *Linear Algebra and Appl.* **1** (1968) 409.

[14] T.-L. Wang and W.B. Gragg, *Math. Comp.* **71** (2002) 1473.

[15] Wikipedia contributors, `https://en.wikipedia.org/w/index.php?title=Chebyshev_polynomials&oldid=999634704`, 2021.