ks_eigensolve_QUDA()

Hwancheol Jeong [*1]

[1]Indiana University, Bloomington, IN 47405, USA

July 3, 2021

Abstract

ks_eigensolve_QUDA() is a routine that calculates eigenvalues and eigenvectors of staggered Dirac operators using Lanczos eigensolvers in the QUDA library. These eigensolvers use the thick restarted Lanczos algorithm with Chebyshev acceleration. A summary of the algorithm and usage of the routine are described.

Contents

1 Introduction

I added a routine named ks_eigensolve_QUDA() in the MILC code [1]. Its definition is in generic_ks/eigen_stuff_QUDA.c. It calculates eigenvalues and eigenvectors of staggered

[*]sonchac@gmail.com

Dirac operators. It calls `eigensolveQUDA()` function from QUDA library [2,3]. Currently, QUDA provides three kinds of eigensolvers: Implicitly Restarted Arnoldi method (`IRAM`), Thick Restarted Lanczos method (`TRLM`), and Thick Restarted Block Lanczos method (`BLKTRLM`). Among them, latter two Lanczos eigensolvers are implemented in `ks_eigensolve_QUDA()`. Both utilize the thick restarted Lanczos algorithm and Chebyshev polynomial acceleration.

2 Lanczos iteration method

Lanczos algorithm is a preconditioning method for eigenvalue problems [4]. It transforms a Hermitian matrix H into a tridiagonal matrix T by a unitary transformation composed of basis vectors of H's Krylov subspace, so called Lanczos vectors. As a unitary transformation, it preserves the eigenvalue spectrum. The most benefit of this preconditioning is that a submatrix of T has eigenvalues approximate to some eigenvalues of H. The Lanczos algorithm iterates constructing each column of T one by one along with an orthogonal basis vector of the Krylov subspace. As the submatrix gets larger by continuing the iteration, its eigenvalues converge to the exact eigenvalues of H. They converge to the largest, smallest, or least dense eigenvalues first.

With a large enough submatrix of T, we compute its eigenvalues and eigenvectors by using a general eigensolver algorithm (or diagonalization algorithm). Both `TRLM` and `BLKTRLM` use Eigen library's `SelfAdjointEigenSolver::compute()` routine for the diagonalization [5], which uses QR iteration method.[1] It is straightforward to calculate eigenvector estimates of H from eigenvectors of T. Their convergences can be determined by error of eigenvalue equation for H.

3 Thick restarted Lanczos

In the standard Lanczos iteration, we have to enlarge the size of the submatrix and so of the Krylov subspace, which means continuing the iteration, until we achieve the wanted precision of eigenvalues and eigenvectors. As the submatrix and Krylov subspace get larger, they demand more system memory to keep eigenvectors and more computations for diagonalization.

Thick restarted (or thick restart) Lanczos algorithm [6] allows us to improve the convergence of eigenvalues while restraining the size of the Krylov subspace. After some number of Lanczos iterations are performed, it restarts the procedure from a smaller Krylov subspace while keeping some (nearly) converged eigenvectors. When restarting, it rotates the Krylov subspace's basis by (nearly) converged k eigenvectors of T, by which the rotated

[1] In fact, `Eigen::SelfAdjointEigenSolver::compute()` also tries to tridiagonalize before starting the QR iteration.

$k \times k$ submatrix becomes diagonal. Resuming the iteration with these rotated basis excludes those (nearly) converged eigenvectors from the newly generated basis vectors. This improves the convergence of remaining eigenvalues. It has known that choosing k to be somewhat higher than the number of well converged eigenvectors, which means keeping a *thick* eigenvector set, gives a better convergence.

4 Polynomial acceleration

A polynomial of matrix transforms the eigenvalue spectrum accordingly, but does not alter eigenvectors. This means we can control the density of eigenvalues for the same set of eigenvectors. Applying a proper polynomial can make the Lanczos iteration converge faster to the wanted eigenvalues and slower to the unwanted eigenvalues, which means unwanted eigenvalues do not come out in the early stages of the iteration. Note that we can regain the eigenvalues of the original matrix from their corresponding eigenvectors by computing their Rayleigh quotients, or Ritz values.

Chebyshev polynomial is a popular choice for this purpose [7, 8]. This function is bounded within $[-1, 1]$ while quickly blows out outside of the region. By a linear transformation, we can map unwanted eigenvalues into the dense bounded region while wanted eigenvalues into the divergent outside region. In other words, the unwanted eigenvalues will be accumulated with a high density, while the wanted eigenvalues will have a very low density. Therefore, it will not only suppress the unwanted eigenvalues but also improve the convergence of wanted eigenvalues.

5 Algorithm details

Let D_s be a staggered Dirac operator. `ks_eigensolve_QUDA()` calculates eigenvalues and eigenvectors of $D_s^\dagger D_s$, which is Hermitian and semi-positive definite, thus its eigenvalues are non-negative real numbers. In addition, the eigenvalue equation of $D_s^\dagger D_s$ can be divided into even parity site part and odd parity site part by a usual even-odd preconditioning. It is enough to perform the Lanczos iteration for the one of even or odd site eigenvalue equation, and then, the other parity solution (eigenvector) can be computed directly by applying D_s to the calculated solution (eigenvector) with a phase difference. This phase difference does not affect our physics observation.

Since $D_s^\dagger D_s$ has only positive[2] eigenvalues, and we are interested in only low-lying eigenmodes, we apply the Chebyshev polynomial so that it maps eigenvalues larger than the largest wanted eigenvalues into the dense bounded region and small eigenvalues into the diverging outer region.

Suppose we want smallest w eigenvalues. We run the Lanczos iteration until m ($> w$) iterations, and calculates eigenvalues and eigenvectors of the $m \times m$ tridiagonal matrix

[2] We cannot have exact zero modes.

using a diagonalization algorithm. After sorting the eigenvalues, we decide smallest k ($\leq m$) eigenvalues to keep with a criterion — converged eigenvalues number + some thick number — and thick-restart the Lanczos iteration from $k+1$ dimensional (rotated) Krylov subspace. In the meantime, if an eigenvector is converged to a machine precision, we lock and exclude it from the future Lanczos iteration. It is possible because the tridiagonal matrix T becomes diagonal at restart. We repeat this procedure until smallest w eigenvalues converge to the target precision.

6 Usage

To use `ks_eigensolve_QUDA()`, the MILC code should be compiled with the QUDA library by turning on `WANTQUDA` and `WANT_EIG_GPU` flags in `Makefile` as well as setting other related variables, such as `QUDA_HOME`, properly.

As other `ks_eigensolver` routines, `ks_eigensolve_QUDA()` gets four input parameters: `eigVec`, `eigVal`, `eigen_param`, and `init`. The last parameter is a dummy variable for compatibility. `eigVec` is an array of fermion field vectors where eigenvectors will be stored, and `eigVal` is an array of real numbers where eigenvalues will be stored. Memories for both of them should be allocated in advance. `eigen_param` is a pointer variable of the `ks_eigen_param` structure, which is defined in `include/imp_ferm_links.h`. All structure members should be appropriately assigned except `Nvecs_in`.

Member variables of `eigen_param` determine the behavior of the Lanczos iteration. `Nvecs` sets the number of wanted eigenvalues. The iteration restarts at `Nkr` Lanczos iterations. `Nkr` must be larger than (`Nvecs` + 4). `blockSize` determines the block size for the thick restarted block Lanczos. `blockSize` = 1 calls the thick restarted Lanczos routine and `blockSize` > 1 calls thick restarted block Lanczos routine. `blockSize` must divides both `Nkr` and `Nvecs`.

`poly` defines a Chebyshev polynomial. `poly.norder` variable defines the order of the Chebyshev polynomial. `poly.minE` and `poly.maxE` are boundary points (eigenvalues) of the transformed bounded region. In other words, eigenvalues between `poly.minE` and `poly.maxE` will not come out in the early iterations. A simple but performance efficient choice of those parameters are setting `poly.minE` to a bit larger than the largest wanted eigenvalue and `poly.maxE` to a bit larger than the largest eigenvalue among the entire spectrum. If `poly.maxE` is set to 0 or a negative value, QUDA will set it automatically to 1.1 times of a largest eigenvalue estimate computed by power iteration.

7 Application example - `ks_eigen_hisq`

`ks_eigen/ks_eigen_hisq` appplication uses the `ks_eigensolve_QUDA()` routine if compiled with `WANTQUDA` and `WANT_EIG_GPU` in `Makefile` turned on.

ks_eigen_hisq	**eigen_param**
Max_Lanczos_restart_iters	MaxIter
eigenval_tolerance	tol
Lanczos_max	Nkr
Chebyshev_alpha	poly.minE
Chebyshev_beta	poly.maxE
Chebyshev_order	poly.norder
block_size	blockSize

Table 1: Correspondence between input parameters of `ks_eigen_hisq` and `eigen_param`.

A test job can be run by '`make check`' command under the `ks_eigen` directory. A sample input file for the test is available at `ks_eigen/test/su3_eigen_hisq.QUDA.2.sample-in`. Dashed lines block input parameters for the Lanczos iteration. These parameters correspond to members of `eigen_param` as in Table 1.

References

[1] `https://github.com/hwancheolJeong/milc_qcd/tree/feature/staggLanczos`.

[2] M. A. Clark, R. Babich, K. Barros, R. C. Brower and C. Rebbi, *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs*, *Comput. Phys. Commun.* **181** (2010) 1517 [`0911.3191`].

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

[4] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, *J. Res. Natl. Bur. Stand. B Math. Sci.* **45** (1950) 255.

[5] G. Guennebaud, B. Jacob et al., "Eigen v3." http://eigen.tuxfamily.org, 2010.

[6] K. Wu and H. Simon, *Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems*, *SIAM J. Matrix Anal. Appl.* **22** (2000) 602.

[7] Y. Saad, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, *Math. Comp.* **42** (1984) 567.

[8] Wikipedia contributors, "Chebyshev polynomials — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Chebyshev_polynomials&oldid=999634704, 2021.