Chebyshev filtering for Abinit users

Antoine Levitt

August 20, 2014

Contents

1 When to use and not to use Chebfi

Chebyshev filtering (Chebfi) is a method to solve the linear eigenvalue problem, and can be used as a SCF solver in Abinit. It was proposed for use in DFT by Zhou et al. [ZSTC06], and implemented in Abinit by Levitt and Torrent [LT14].

The design goal is for Chebfi to replace LOBPCG as the solver of choice for large-scale computations in Abinit. By performing less orthogonalizations and diagonalizations than LOBPCG, scaling to higher processor counts is possible (see the experiments in [LT14]).

2 How to use Chebfi

Simply set `wfoptalg` to 1, and set the `np*` variables, as in LOBPCG. In particular, begin by setting `npkpt` and `npspinor` to the maximum value possible: the tasks are mostly independent and the speedup nearly optimal. As a starting point, for large processor counts, use `npfft` $\approx$ `npband`, and `np_slk` = `nband`/10.

3 Convergence

Mostly due to the absence of a preconditionner in Chebfi, the convergence is sometimes worse than with LOBPCG. In some cases the difference is unnoticeable, in others it might be uncompetitively slow: try for yourself on your system! When convergence is poor, it is usually a good idea to use more bands than strictly necessary, by increasing `nband`. This increases the cost per iteration but improves convergence: a trade-off is needed. Note that the last bands will always converge very slowly or not at all, by design: use `nbdbuf` to discard these when computing the wavefunction residual.

4 Chebfi vs LOBPCG

Chebfi is usually faster per iteration than LOBPCG, up to factors of about 5 for high processor counts. It is much less prone to instabilities and spurious NaNs. It is easier to tune: the options `bandpp` and `use_slk` are not used, and the convergence behaviour does not depend on the parallelization - the results are the same than with the serial version). On the other hand, for some systems, Chebfi might be very slow to converge.

5 Performance

5.1 Environment

For good performance it is *imperative* to use optimized linear algebra libraries. On Intel systems, MKL is a good choice. On other systems, use the vendor-provided library, or good free ones such as OpenBLAS. Never use the BLAS/LAPACK fallback in Abinit.

Any good compiler with a reasonable level of optimization (gfortran or ifort with O2 or greater) should be fine. You should also have a good MPI implementation.

For large computations, you need to have ScaLAPACK installed. You should also use the optional ELPA library to speed up the diagonalizations.

FFTs are usually faster with the FFTW library: see `fftalg`.

5.2 Measures

Use the `timopt` variable to print a breakdown of time spent at the end of the output file. If available, PAPI (`papiopt`) might give more precise measurments. You are looking for the bit that looks like

```
 Partitioning of chebfi
- chebfi                        45.939  34.0     48.072  29.0            48

- chebfi(getghc)                27.306  20.2     28.015  16.9           240
- chebfi(opernla)                3.505   2.6      3.583   2.2           192
- chebfi(opernlb)                3.055   2.3      3.124   1.9           192
- chebfi(inv_s)                  2.218   1.6      2.273   1.4           192
- chebfi(alltoall)               0.261   0.2      0.260   0.2            96
- chebfi(rotation)               2.089   1.5      2.146   1.3            48
```

```
- chebfi(subdiago)              1.552   1.2      2.596   1.6            48
- chebfi(subham)                1.000   0.7      1.024   0.6            48
- chebfi(residuals)             0.239   0.2      0.242   0.1            48
- chebfi(update_eigens)         0.226   0.2      0.220   0.1            48
- chebfi(sync)                  3.045   2.3      3.068   1.9            96
```

The interesting part is the third column, that gives the percent of total time spent in specific routines. The code scales well (is not limited by communications) when `getghc`,`opernla`, `opernlb` and `inv_s` dominate. It has stopped scaling when the communications dominate : `alltoall`, `subdiago` and `subham`. A large `sync` is usually the sign of a suboptimal MPI implementation.

Also interesting is the breakdown of `getghc`, that gives the time spent in Fourier and nonlocal operator applications.

```
 Partitioning of getghc
- getghc                       26.335  19.5     27.025  16.3           240

- fourwf%getghc                15.291  11.3     15.692   9.5           240
- nonlop%getghc                10.849   8.0     11.138   6.7           240
- getghc-other                  0.195   0.1      0.195   0.1           -12
```

For large computations, the time spent in the FFT (fourwf) operations should be small.

6 Tuning and trade-offs

6.1 `nline`

This option controls the number of applications of the Hamiltonian per band per iteration. If it is too small, Rayleigh-Ritz steps are too frequent, which degrades parallel scaling. If it is too large, too much time is spent optimizing wavefunctions, while the density is not converged. Don't increase it too much (above 10), or instabilities will occur.

6.2 `np_slk`

`np_slk` is the number of processors involved in ScaLAPACK calls. It might be interesting to set this to a lower value than the total number of processors, as diagonalization stops scaling well before the rest of the code.

Try varying this and monitoring the `subdiago` metric. If the time spent in `subham` is too high, try reducing `np_slk`. A characteristic value is around `nband/10`, but your mileage may vary.

6.3 `npfft` vs `npband`

For large computations, it is usually a good idea to use as large a value of `npfft` as possible (the maximum value is limited by the size of the FFT grid). If the time spent in `fourwf` is too large, decrease `npfft`.

6.4 `use_gemm_nonlop`

This might improve substantially the computation of the nonlocal part of the Hamiltonian (`nonlop`). Its usefulness increases with the number of bands treated by processor, ie `nband/npband`.

6.5 Memory

Some arrays are distributed on `npfft` processors only ; some are distributed on `np_slk` only. If you run out of memory, increasing one of these variables might solve the problem.

References

[LT14] Antoine Levitt and Marc Torrent. Parallel eigensolvers in plane-wave density functional theory. *arXiv preprint arXiv:1406.4350*, 2014.

[ZSTC06] Yunkai Zhou, Yousef Saad, Murilo L Tiago, and James R Chelikowsky. Self-consistent-field calculations using chebyshev-filtered subspace iteration. *Journal of Computational Physics*, 219(1):172–184, 2006.