New implementation of Chebyshev filtering inside ABINIT

B.Sataric\textsuperscript{1}, J.Bieder\textsuperscript{2}, M. Torrent\textsuperscript{3} and W.Jalby\textsuperscript{1}

\textsuperscript{1} University of Versailles UVSQ, France
\textsuperscript{2} University of Liege, Belgium
\textsuperscript{3} CEA, DAM, DIF, Arpajon, France
Chebyshev filtering algorithm

- Eigenvalues of Eigenproblem $H\psi = \lambda S\psi$ can be represented by $\Lambda$, and eigenvectors can be represented by $P$
- In that case eigenproblem notation becomes $HP = SPA$ or $S^{-1}H = P\Lambda P^{-1}$
- Spectral filter $T_n$ can be used to filter eigencomponents as given in formula: $T_n (S^{-1} H)\psi = PT_n (\Lambda) P^{-1} \psi$
- Rayleigh-Ritz procedure is used to separate the individual eigenvectors and eigenvalues, and iterate until convergence
Chebyshev filtering algorithm

- Input: a set of $N_{pw} \times N$ bands wavefunctions $\Psi$
- Output: the updated wave-functions $\Psi$
  - Locate eigenvalue spectrum
    - Compute Rayleigh quotients for every band, and set $\lambda_*$ equal to the largest one
    - Set $\lambda_*$ to be an upper bound of the spectrum
    - Compute the filter center and radius $c = (\lambda_* + \lambda_*/2$, $r = (\lambda_* - \lambda_*)/2$
  - Compute Chebyshev polynomial for each eigenvector
    - for each band $\psi$ do
      - Set $\psi^0 = \psi$, and $\psi^1 = 1/r * (S^{-1} H\psi^0 - c\psi^0 )$
      - for $i = 2, \ldots, n_{inner}$ do
        - $\psi^i = 2/r * (S^{-1} H\psi^{i-1} - c\psi^{i-1} ) - \psi^{i-2}$
      - end for
    - end for
  - Apply Rayleigh-Ritz procedure
    - Compute the subspace matrices $H_\psi = \Psi^T H \Psi$, and $S_\psi = \Psi^T S \Psi$
    - Solve the dense generalized eigenproblem $H_\psi X = S_\psi X \Lambda$, where $\Lambda$ is a diagonal matrix of eigenvalues, and $X$ is the $S_\psi$-orthonormal set of eigenvectors
    - Do the subspace rotation $\Psi \leftarrow \Psi X$
Abinit abstract layer (xg datatypes – developed by Jordan Bieder$^2$)

- Highly efficient multi-threaded wrapper module for BLAS/LAPACK (level-1 and level 2) routine calls
- Module is used to help developer use 2D arrays and their subblocks with ease (by xgBlock pointer objects)
- It contains sub-module used for MPI matrix transpositions (all-to-all and all-gether)
- New functions added during Chebfi2 development:
  - xgBlock_colwiseDivision
  - xgBlock_saxpy
- XG provided smooth translation of CB1 code into CB2 without worrying about particular details of BLAS or LAPACK function parameters, Fortran pointers or OpenMP pragmas and variables
Xg usage example (Chebfi 2 nextOrderPolynom)

if (chebfi%paw) then
   !apply matrix inverse function
   call getBm1X(chebfi%xAXColsRows, chebfi%X_next)
else
   !copy xAXColsRows into X_next array
   call xgBlock_copy(chebfi%xAXColsRows, chebfi%X_next, 1, 1)
end if

!scale xXColsRows by center
call xgBlock_scale(chebfi%xXColsRows, center, 1)
!important X_next = X_next - xXColsRows
!scale xXColsRows by 1/center
call xgBlock_scale(chebfi%xXColsRows, 1/center, 1)

if (iline == 0) then
   !scale X_next by 1/radius
   call xgBlock_scale(chebfi%X_next, one_over_r, 1)
else
   !scale X_next by 2/radius
   call xgBlock_scale(chebfi%X_next, two_over_r, 1)
   !X_next = X_next - X_prev
   call xgBlock_saxpy(chebfi%X_next, dble(-1.0), chebfi%X_prev)
end if
Different solver scaling on Intel Xeon Cascadelake
TODO list and expectations

• TODO:
  • Finalization of MPI transposition
  • Optimization of coding (Hamiltonian application and inverse matrix calculation)
  • Addition of nspinors=2 capability
  • Automation of task distribution

• Expectations:
  • Better MPI scaling of Chebfi2 than LOBPCG2 because Rayleigh-Ritz procedure is done only once (instead of once per iteration) – thus reducing communication
  • Chebfi2 will be available for use as a standalone library