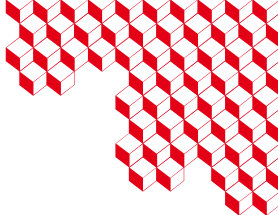




université
PARIS-SACLAY



Iterative eigensolvers in Abinit

Clémentine Barat, Marc Torrent

CEA, DAM, DIF

Université Paris-Saclay, CEA, Laboratoire Matière en Conditions Extrêmes

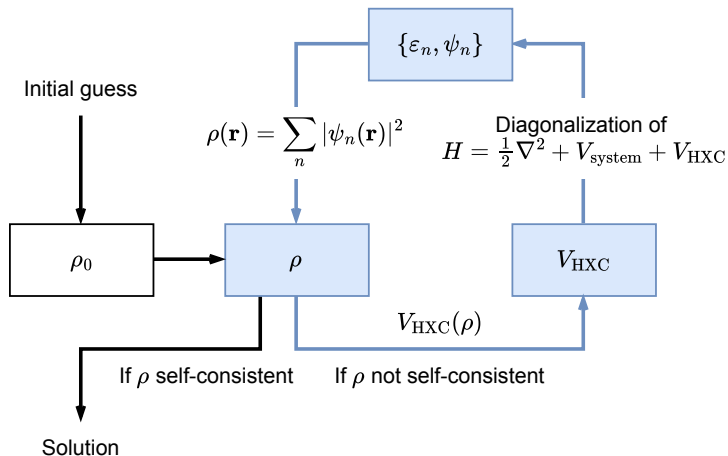
LMO, Université Paris-Saclay

Feb. 4, 2026



DFT parallelization strategy

Self-consistent field in DFT



Parallelization possibilities in plane-wave DFT

$$\rho(\mathbf{r}) = \sum_{\alpha \in \{\uparrow, \downarrow\}} \sum_{\mathbf{k} \in \text{BZ}_M} \sum_{n=1}^m |\psi_n^{k, \alpha}(\mathbf{r})|^2$$

| Parallelization level | spins | k-points | bands |
|-----------------------|--------|---------------|-------------|
| System-size scaling | 2 | $\propto 1/S$ | $\propto S$ |
| Parallel speedup | linear | linear | ? |

S: system size



Iterative diagonalization algorithms

Eigsolver in SCF iterations

The diagonalization is the computational bottleneck of the SCF iterations.

- Iterative, matrix-free eigensolver
- SCF and eigensolver iterations mixed

Algorithm 1 SCF iterations with an exact eigensolver

```
 $\rho \leftarrow \text{GUESSDENSITY}$   
while  $\rho$  not converged do  
   $E, \Psi \leftarrow \text{EIGENSOLVER}(H[\rho])$   
   $\rho \leftarrow \text{MAKEDENSITY}(\Psi)$   
end while
```

Algorithm 2 SCF iteration with an iterative eigensolver

```
 $\Psi \leftarrow \text{GUESSVECTORS}$   
 $\rho \leftarrow \text{MAKEDENSITY}(\Psi)$   
while  $\rho$  not converged do  
   $E, \Psi \leftarrow \text{EIGENSOLVERSTEP}(H[\rho], \Psi)$   
   $\rho \leftarrow \text{MAKEDENSITY}(\Psi)$   
end while
```

The diagonalization problem in DFT

In KSDFT we want to find the m smallest eigenpairs of H , $(\varepsilon_1, \psi_1), \dots, (\varepsilon_m, \psi_m)$ in $\mathbb{R} \times \mathbb{C}^N$ s.t. for $1 \leq i \leq m$

$$H\psi_i = \varepsilon_i\psi_i$$

m : Number of bands

N : Number of plane-waves

H : Discretized Hamiltonian

Two main families of iterative eigensolvers are implemented in Abinit to solve this problem:

- Minimization algorithms: An iterative minimization method is used to solve the equivalent minimization problem

$$\begin{aligned} \min_{X \in \mathbb{C}^{N \times m}} \|X^H H X\| \\ \text{s.t. } X^H X = I_m \end{aligned}$$

- Filtering algorithms: The eigensubspace $\text{Span}\{\psi_1, \dots, \psi_m\}$ is approximated iteratively with a filter.

The Rayleigh-Ritz method

The Rayleigh-Ritz method gives an approximation of the eigenvalues and eigenvectors of a matrix projected in a given subspace.

Algorithm 3 Rayleigh-Ritz

function RAYLEIGHRITZ(H, X)

$X \leftarrow \text{ORTHO}(X)$

$R \leftarrow X^H H X$

$E, Y \leftarrow \text{EIGEN}(R)$

$\Psi \leftarrow X Y$

return E, Ψ

end function

- $$\overset{\overbrace{\quad m \quad}}{\boxed{R}} = \boxed{X^H} \cdot \overset{\overbrace{\quad N \quad}}{\boxed{H}} \cdot \boxed{X}$$
- E, Y full diagonalization of $R \longrightarrow \mathcal{O}(m^3)$
- $$\boxed{\Psi} = \boxed{X} \cdot \boxed{Y}$$

Minimization algorithms for eigenvalue problems

- Eigenvalue problem: Find the m smallest eigenpairs $(\varepsilon_1, \psi_1), \dots, (\varepsilon_m, \psi_m)$ in $\mathbb{R} \times \mathbb{C}^N$ such that for $1 \leq i \leq m$,

$$H\psi_i = \varepsilon_i\psi_i \quad (1)$$

- Constrained minimization problem:

$$\begin{aligned} \min_{X \in \mathbb{C}^{N \times m}} \quad & \|X^H H X\| \\ \text{s.t.} \quad & X^H X = I_m \end{aligned} \quad (2)$$

$\|\cdot\|$ is the operator norm

The solutions of (2) spans the same subspace as the solutions of (1).

Standard conjugate gradient algorithm for quadratic minimization problems

Quadratic optimization problem

$$\min_{x \in \mathbb{R}^N} q(x) = \frac{1}{2} x^H A x - b^H x - x^H b$$

$A \in \mathbb{C}^{N \times N}$ hermitian positive definite
 $b \in \mathbb{C}^N$

q can be independently minimized on a set of A -conjugated directions $(d_1, \dots, d_N$ with $d_i^T A d_j = 0$ for $i \neq j$).

$$q(\alpha d_i + \beta d_j) = \alpha q(d_i) + \beta q(d_j)$$

Algorithm 4 Conjugate gradient

$x \leftarrow \text{GUESS}$

$d \leftarrow -g(x)$ ▷ *line search direction*

▷ $g(x) = \nabla q(x) = Ax - b$

while x not converged **do**

$\alpha \leftarrow \text{ARGMIN}(\alpha \mapsto q(x + \alpha d))$

$x \leftarrow x + \alpha d$

▷ *minimize q along d*

$d \leftarrow \text{CONJUGATE}(-g(x), d)$

▷ $d \leftarrow -g(x) + \beta d$ such that d is
 A -conjugate to previous directions

end while

Conjugate gradient algorithm for eigenvalue problems

Eigenvalue problem

$$\min_{x \in \mathbb{C}^N} \varepsilon(x) = \frac{x^H H x}{x^H x}$$

$H \in \mathbb{C}^{N \times N}$ hermitian

Locally around a minimizer, ε behaves like a quadratic function with gradient

$$g(x) = \frac{2}{\|x\|} (H - \varepsilon(x)I)x$$

Algorithm 5 Conjugate gradient for eigenvalue problems

$x \leftarrow \text{GUESS}$

$d \leftarrow -g(x)$

▷ *Now $g(x) = 2/\|x\|(H - \varepsilon(x)I)x$*

while x not converged **do**

$x \leftarrow \text{RAYLEIGHRITZMIN}(H, \text{Span}\{x, d\})$

▷ *the minimization of ε along d is a 2×2 eigenvalue problem*

3-terms variant :

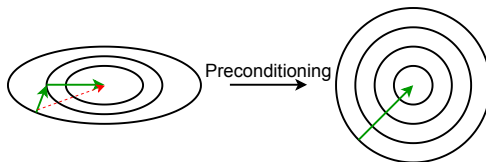
$x \leftarrow \text{RAYLEIGHRITZMIN}(H, \text{Span}\{x, d, x^{\text{old}}\})$

$d \leftarrow \text{CONJUGATE}(-g(x), d)$

▷ *$d \leftarrow -g(x) + \beta d$ with a β that "conjugates" d to the previous direction*

end while

Preconditioning



Ideally, we would have the search direction directly proportional to the error $x - \psi_1$.

Replace the search direction

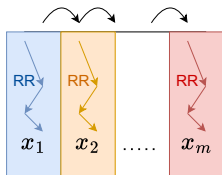
$$d(x) = -g(x) \quad \rightarrow \quad d(x) = -\mathbf{T}g(x)$$

with T an invertible matrix such that $Tg(x) \approx \alpha(x - \psi_1)$ (when x is close to the solution).

Projected conjugate gradient algorithm

Eigenvalue problem in KSDF

$$\begin{aligned} \min_{X \in \mathbb{C}^{N \times m}} E(X) &= \|X^H H X\| \\ \text{s.t. } X^H X &= I_m \end{aligned}$$



Algorithm 6 Projected conjugate gradient for KSDF

for $i = 1$ to m **do**

$x_i \leftarrow$ GUESS

$x_i \leftarrow$ ORTHOGONALPROJ($x_i, \{x_1, \dots, x_{i-1}\}$)

$d \leftarrow$ ORTHOGONALPROJ($-Tg(x_i), \{x_1, \dots, x_{i-1}\}$)

▷ *line search direction d and initial guess orthogonal to previously computed directions*

while x_i not converged **do**

$x_i \leftarrow$ RAYLEIGHRITZMIN($H, \text{Span}\{x_i, d, x_i^{\text{old}}\}$)

$d \leftarrow$ CONJUGATE($-Tg(x_i), d$)

$d \leftarrow$ ORTHOGONALPROJ($d, \{x_1, \dots, x_{i-1}\}$)

end while

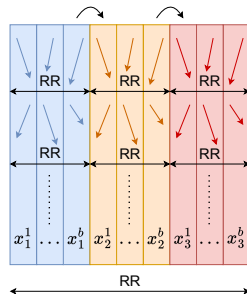
end for

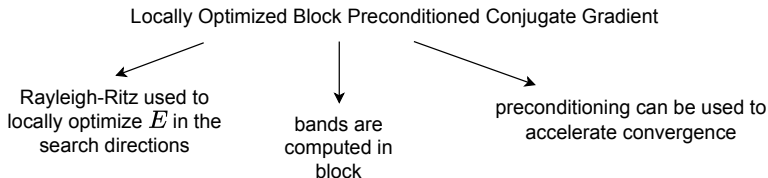
The $\{x_i\}$ directly approximate the eigenvectors $\{\psi_i\} \rightarrow$ no final Rayleigh-Ritz needed

LOBPCG: Locally Optimized Block Preconditioned Conjugate Gradient

Algorithm 7 LOBPCG

```
for  $i = 1$  to  $n_b$  do ▷ number of blocks =  $n_b$   
   $X_i = (x_i^1 | \dots | x_i^b) \leftarrow \text{GUESS}$  ▷  $X_i$  of size  $N \times b$  with  $b = \text{block size}$   
   $X_i \leftarrow \text{ORTHOGONALPROJ}(X_i, \{X_1, \dots, X_{i-1}\})$   
  for  $j = 1$  to  $b$  do  
     $d_j \leftarrow \text{ORTHOGONALPROJ}(-Tg(x_i^j), \{X_1, \dots, X_{i-1}\})$   
  end for ▷ highly parallelizable  
  while  $X_i$  not converged do  
     $X_i \leftarrow \text{RAYLEIGHRITZMIN}(H, \text{Span}\{X_i, X_i^{\text{old}}, d_1, \dots, d_b\})$   
    ▷ 3 term variant → RayleighRitz in dimension  $3b$   
    for  $j = 1$  to  $b$  do  
       $d_j \leftarrow \text{CONJUGATE}(-Tg(x_i^j), d_j)$   
       $d_j \leftarrow \text{ORTHOGONALPROJ}(d_j, \{X_1, \dots, X_{i-1}\})$   
    end for ▷ highly parallelizable  
  end while  
end for  
 $\psi_1, \dots, \psi_m \leftarrow \text{RAYLEIGHRITZ}(X_1, \dots, X_{n_b})$ 
```





- Parallelization possibilities in each block: The b new search directions can be computed in parallel
- One Rayleigh-Ritz in dimension $3b$ per iteration, per block
- **A balance must be found between the parallelization of the Hamiltonian application and the dimension of the Rayleigh-Ritz**
- One final Rayleigh-Ritz in dimension m for stability

Filtering methods

In subspace Filtering methods, we approximate directly the subspace $\mathcal{U} = \text{Span}\{\psi_1, \dots, \psi_m\}$.

- Initialization : Initial (random) subspace of dimension m :

$$\mathcal{X}_0 = \text{Span}\{x_1^0, \dots, x_m^0\}$$

- Iterations : Apply a filter $f : \mathbb{C}^N \rightarrow \mathbb{C}^N$ well chosen:

$$\mathcal{X}_{k+1} = f(\mathcal{X}_k) = \text{Span}\{f(x_1^k), \dots, f(x_m^k)\}$$

- ideally, f is the orthogonal projector on \mathcal{U} such that $\mathcal{X}_1 = f(\mathcal{X}_0) = \mathcal{U}$.
- in practice, we use polynomial filters $f = P(H)$ with P a polynomial:

$$f(x) = P(H)x = \sum_{i=0}^N P(\varepsilon_i) \langle \psi_i, x \rangle \psi_i$$

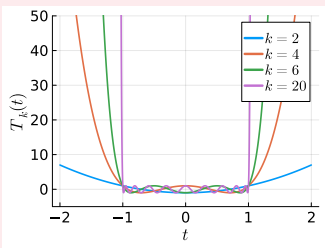
→ we want P to be large on $[\varepsilon_1, \varepsilon_m]$ and small on $[\varepsilon_{m+1}, \varepsilon_N]$.

- Final step: Apply the Rayleigh-Ritz method to retrieve the eigenvectors from the subspace estimation

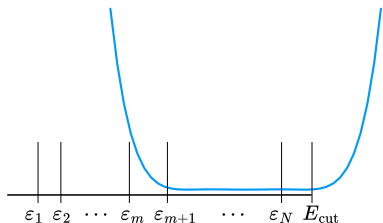
Chebyshev polynomials as filters: ChebFi

Chebyshev polynomials of the first kind T_k

- $T_k(x) \in [-1, 1]$ for $x \in [-1, 1]$
- T_k grows rapidly outside of $[-1, 1]$



Chebyshev polynomials are used as filter, by translating the unwanted part of the spectrum ($[\varepsilon_{m+1}, \varepsilon_N]$) to $[-1, 1]$.



Algorithm 8 ChebFi

```
 $\Psi \leftarrow \text{GUESSVECTORS}$   
 $\rho \leftarrow \text{MAKEDENSITY}(\Psi)$   
while  $\rho$  not converged do  
   $H \leftarrow H[\rho]$   
   $X \leftarrow F_{\text{cheby}}(H)(\Psi)$   
     $\triangleright$  the filter can be applied in parallel to  
      each vector  
     $\triangleright$  degree of the Chebyshev polynomial  
      = number of Hamiltonian application  
   $E, \Psi \leftarrow \text{RAYLEIGHRITZ}(H, X)$   
   $\rho \leftarrow \text{MAKEDENSITY}(\Psi)$   
end while
```

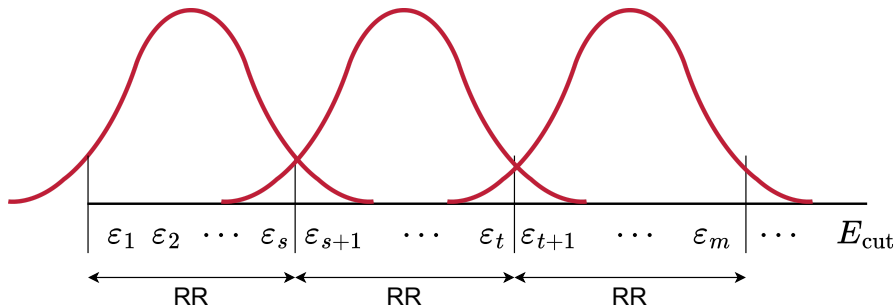
Notes :

- Parallelization possibilities: The filter can be applied in parallel to each vector.
- One Rayleigh-Ritz in dimension m per SCF iteration
- The last bands converge more slowly
→ compute more bands than necessary

To come: Spectrum Slicing

Eigensubspace $\text{Span}\{\psi_1, \dots, \psi_m\}$ is split into slices that are independently filtered.

One Rayleigh-Ritz in smaller dimension per slice



Comparison of LOBPCG and ChebFi

| | LOBPCG | ChebFi |
|------------------------|---|---|
| Cost per SCF iteration | $\mathcal{O}(n_{\text{block}} \cdot n_{\text{line}} \cdot (T_H \cdot \frac{m}{n_{\text{block}}} + (3 \frac{m}{n_{\text{block}}})^3) + m^3)$ | $\mathcal{O}(n_{\text{deg}} \cdot T_H \cdot m + m^3)$ |

terms in red can be reduced to 1 with parallelization

T_H : cost of one Hamiltonian application



Parallelism with ABINIT

How to ...

Using MPI parallelism with Abinit

■ Job

Number of MPI processes `#MSUB -n <num_procs>`
 `mpirun -n <num_procs> ...`

■ Abinit input file

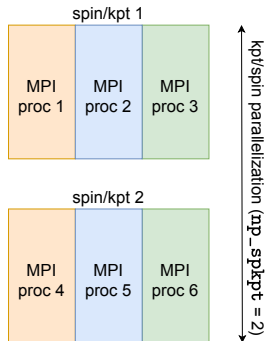
`paral_kgb` 0 : spin/k-point parallelization only
 1 : spin/k-point & band parallelization

`np_spkpt` number of MPI processes for
 k-point/spin parallelization

`npband` number of MPI processes for
 band parallelization

Total number of MPI processes
 $\langle \text{num_procs} \rangle = \text{np_spkpt} \times \text{npband}$

band parallelization (**npband** = 3)



Guidelines: First parallelize over spins and k-points, then over bands

Using OpenMP parallelism with Abinit

■ Job

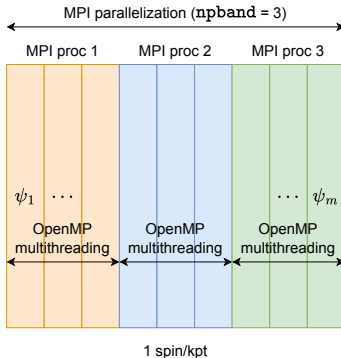
Number of threads `#MSUB -c <num_threads>`

■ Environment variables

```
export OMP_NUM_THREADS=<num_threads>
```

Guidelines:

Hybrid (MPI+OpenMP) band parallelization is more efficient.
`<num_thread>` should be smaller or equal to the number of core per node.



Using GPU parallelism with Abinit

■ Abinit input file

`gpu_option` 0 or GPU_DISABLED : no use of GPU
 2 or GPU_OPENMP : use GPU (OPENMP_OFFLOAD programming model)

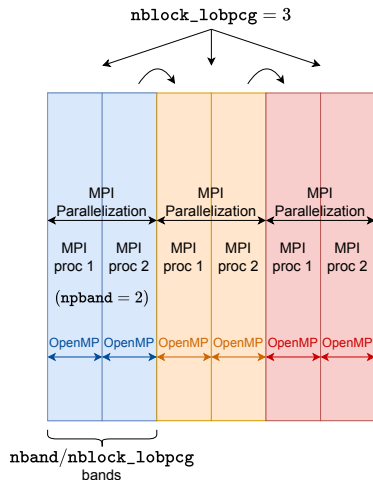
Guidelines:

Use one MPI process per GPU & no OpenMP multithreading.

Using LOBPCG in Abinit

- `wfoptalg` : set to 114 to select LOBPCG.
- `nblock_lobpcg` : number of blocks in LOBPCG
- `nline` : number of local searches (Rayleigh-Ritz) per SCF step.

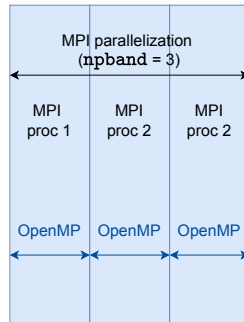
$\text{nband}/\text{nblock_lobpcg}$ should be a multiple of $\text{npband} \times \langle \text{num_threads} \rangle$ for an optimal band parallelization.



Using ChebFi in Abinit

- `wfoptalg` : set to 111 to select ChebFi
- `mdeg_filter` : maximum degree of the Chebyshev polynomial
- `nbdbuf` : number of (highest energy) bands whose convergence won't be checked

`nband` should be a multiple of `npband` \times `<num_threads>` for an optimal band parallelization.

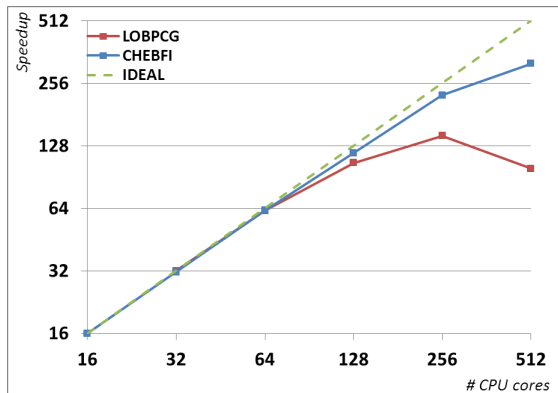




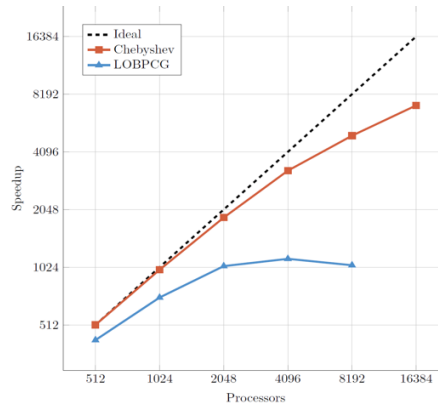
ABINIT parallel performances

LOBPCG vs ChebFi

UO₂ crystal, 98 atoms, 512 bands

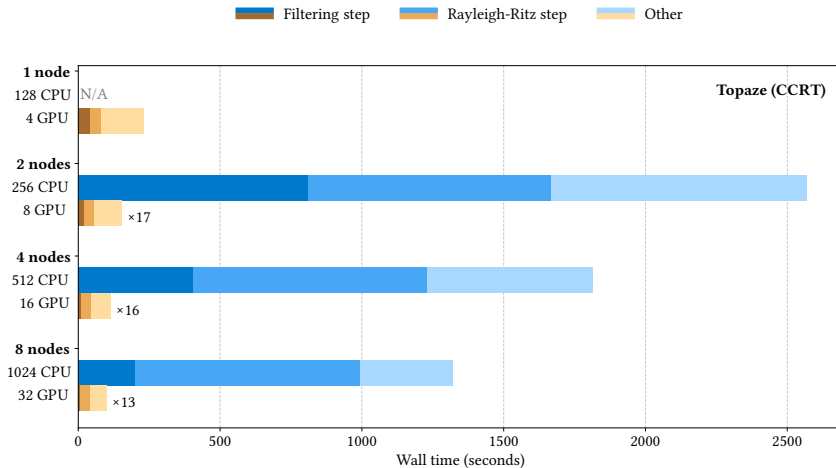


Ti crystal, 256 atoms, 2048 bands



CPU vs GPU

Ti solid-liquid interface, 255 atoms, 4096 bands - ChebFi



Thank you for your attention !