DE LA RECHERCHE À L'INDUSTRIE

Abinit performances for systems with many atoms: Some analyses and perspectives

L. Baguet[1,2], M. Torrent[1,2]

[1] CEA, DAM, DIF, F-91297 Arpajon, France
[2] Université Paris-Saclay, CEA, Laboratoire Matière en Conditions Extrêmes, 91680 Bruyères-le-Châtel, France

Table of contents

Table of contents

Performances dependencies

Performances of Abinit depend on many factors :

- Self-Consistent Field method
- Diagonalization Alg. (CG, LOBPCG, Chebyshev filtering, RMM-DIIS)
- parallelization scheme (Number of MPI processes ? Threads ?)
- libraries used (for FFT and linear algebra)
- compilation options (vectorization, . . .)

In addition to all that :

- best choice depends on the system and the computer architecture
- architectures evolve with time... (new types of CPU, more GPUs soon)

Performances dependencies

Performances of Abinit depend on many factors :

- Self-Consistent Field method
- Diagonalization Alg. (CG, LOBPCG, Chebyshev filtering, RMM-DIIS)
- parallelization scheme (Number of MPI processes ? Threads ?)
- libraries used (for FFT and linear algebra)
- compilation options (vectorization, . . .)

In addition to all that :

- best choice depends on the system and the computer architecture
- architectures evolve with time... (new types of CPU, more GPUs soon)

The only solution to understand these effects is to **measure** performances

Performances dependencies

Performances of Abinit depend on many factors :

- Self-Consistent Field method
- Diagonalization Alg. (CG, LOBPCG, Chebyshev filtering, RMM-DIIS)
- parallelization scheme (Number of MPI processes ? Threads ?)
- libraries used (for FFT and linear algebra)
- compilation options (vectorization, . . .)

In addition to all that :

- best choice depends on the system and the computer architecture
- architectures evolve with time... (new types of CPU, more GPUs soon)

The only solution to understand these effects is to **measure** performances

An example : 255 atoms of Ti

System :

- $N_{at} = 255$ atoms (Ti), 2048 bands
- Computation at Γ : $nkpt = 1$
- cutoff energy : 5 Ha (10 Ha for double grid)

Architecture :

- Intel Skylake (24 cores per processors) 2 processors per node cores
- Total of 256 cores used

Algo	MPI	Threads	npfft	npband	bandpp	Nbloc	T it>3 (s)	Mem (Mb)	Mem/core
LOBPCG	64	4	1	64	16	2	102	1437	359
LOBPCG	32	8	1	32	16	4	29	1176	147
LOBPCG	16	16	1	16	32	4	30	1281	80
Cheb. Fi.	256	1	4	64	x	x	31	737	737
Cheb. Fi.	32	8	1	32	x	x	24	1357	170

Note : “T it$>$3” $=$ time per SCF iteration after the 3^{rd} iteration (rather constant)

An example : 255 atoms of Ti

System :

- $N_{at} = 255$ atoms (Ti), 2048 bands
- Computation at Γ : $nkpt = 1$
- cutoff energy : 5 Ha (10 Ha for double grid)

Architecture :

- Intel Skylake (24 cores per processors) 2 processors per node cores
- Total of 256 cores used

Algo	MPI	Threads	npfft	npband	bandpp	Nbloc	T it>3 (s)	Mem (Mb)	Mem/core
LOBPCG	64	4	1	64	16	2	102	1437	359
LOBPCG	32	8	1	32	16	4	29	1176	147
LOBPCG	16	16	1	16	32	4	30	1281	80
Cheb. Fi.	256	1	4	64	x	x	31	737	737
Cheb. Fi.	32	8	1	32	x	x	24	1357	170

Note : "T it$>$3" $=$ time per SCF iteration after the 3^{rd} iteration (rather constant)

- LOBPCG : better performances when using more threads
- Chebyshev Filtering gives good results too

An example : 255 atoms of Ti

System :

- $N_{at} = 255$ atoms (Ti), 2048 bands
- Computation at Γ : $nkpt = 1$
- cutoff energy : 5 Ha (10 Ha for double grid)

Architecture :

- Intel Skylake (24 cores per processors) 2 processors per node cores
- Total of 256 cores used

Algo	MPI	Threads	npfft	npband	bandpp	Nbloc	T it>3 (s)	Mem (Mb)	Mem/core
LOBPCG	64	4	1	64	16	2	102	1437	359
LOBPCG	32	8	1	32	16	4	29	1176	147
LOBPCG	16	16	1	16	32	4	30	1281	80
Cheb. Fi.	256	1	4	64	x	x	31	737	737
Cheb. Fi.	32	8	1	32	x	x	24	1357	170

Note : "T it>3" = time per SCF iteration after the 3^{rd} iteration (rather constant)

- LOBPCG : better performances when using more threads
- Chebyshev Filtering gives good results too

More investigation is needed $\Rightarrow$ **Plan to use the "hpc tests"**

Table of contents

An other example : 144 SiO_2

System :

- $N_{at} = 144$ atoms (SiO_2), 460 bands
- Computation at Γ : $nkpt = 1$
- cutoff energy : 15 Ha (30 Ha for double grid) : $N_{PW} \approx$ 23 000
- PAW pseudo (so $useylm = 1$), $N_{proj} = 8$ projectors $|p_{a,nlm}\rangle$ per atom

Parameters of the ground state computation (not converged) :

- Conjugate Gradient ($wfoptalg = 10$) : $nstep = 10$, $nnsclo = 1$, $nline = 2$
- sequential : 1 MPI process, 1 thread

With $timopt = -1$:

```
-<BEGIN_TIMER mpi_nprocs = 1, omp_nthreads = 1, mpi_rank = world>
- cpu_time =        2524.0, wall_time =        2524.1
-
- routine                          cpu     %       wall     %
-
- nonlop(apply)                1355.267  53.7   1355.319  53.7
- projbd                        605.167  24.0    605.219  24.0
- vtowfk(ssdiag)                170.550   6.8    170.556   6.8
- vtowfk(contrib)               168.361   6.7    168.363   6.7
- fourwf%(pot)                   93.632   3.7     93.645   3.7
- nonlop(forstr)                 25.037   1.0     25.037   1.0
- fourwf%(den)                   13.132   0.5     13.135   0.5
- newkpt(excl. rwwf   )          10.554   0.4     10.554   0.4
```

Non-local operator

$N_{at} = 144 \;\; N_{proj} = 8 \;\; N_{PW} \approx 23000$

The application of the non-local operator on $|\psi\rangle$ is :

$$H_{\text{nl}}|\psi\rangle = \sum_{a}^{N_{at}} \sum_{ij}^{N_{proj}} |p_{ai}\rangle D_{aij}\, cprj_{aj}$$

$$cprj_{ai} = \langle p_{ai}|\psi\rangle$$

The application of the overlap operator is computed the same way :

$$S|\psi\rangle = |\psi\rangle + \sum_{a}^{N_{at}} \sum_{ij}^{N_{proj}} |p_{ai}\rangle S_{aij}\, cprj_{aj}$$

In Abinit, the computation of $H_{\text{nl}}|\psi\rangle$ (or $S|\psi\rangle$) is done in "nonlop_ylm", in 3 steps :

- "opernla" : $cprj_{ai} = \langle p_{ai}|\psi\rangle$: $O(N_{proj} N_{at} N_{PW})$ operations
- "opernlc" : $f_{ai} = \sum_j D_{aij} cprj_{aj}$: $O((N_{proj})^2 N_{at})$ operations
- "opernlb" : $H_{\text{nl}}|\psi\rangle = \sum_{a,i} |p_{ai}\rangle f_{ai}$: $O(N_{proj} N_{at} N_{PW})$ operations

In practice : $T_{\text{nonlop}} = T_{\text{opernla}} + T_{\text{opernlb}}$. Here : $T_{\text{nonlop}} \approx 0.5 T_{\text{total}}$.

Non-local operator

$N_{at} = 144 \;\; N_{proj} = 8 \;\; N_{PW} \approx 23000$

The application of the non-local operator on $|\psi\rangle$ is :

$$H_{\text{nl}}|\psi\rangle = \sum_{a}^{N_{at}} \sum_{ij}^{N_{proj}} |p_{ai}\rangle D_{aij} \; cprj_{aj}$$

$$cprj_{ai} = \langle p_{ai}|\psi\rangle$$

The application of the overlap operator is computed the same way :

$$S|\psi\rangle = |\psi\rangle + \sum_{a}^{N_{at}} \sum_{ij}^{N_{proj}} |p_{ai}\rangle S_{aij} \; cprj_{aj}$$

In Abinit, the computation of $H_{\text{nl}}|\psi\rangle$ (or $S|\psi\rangle$) is done in "nonlop_ylm", in 3 steps :

- "opernla" : $cprj_{ai} = \langle p_{ai}|\psi\rangle$: $O(N_{proj} N_{at} N_{PW})$ operations
- "opernlc" : $f_{ai} = \sum_j D_{aij} cprj_{aj}$: $O((N_{proj})^2 N_{at})$ operations
- "opernlb" : $H_{\text{nl}}|\psi\rangle = \sum_{a,i} |p_{ai}\rangle f_{ai}$: $O(N_{proj} N_{at} N_{PW})$ operations

In practice : $T_{\text{nonlop}} = T_{\text{opernla}} + T_{\text{opernlb}}$. Here : $T_{\text{nonlop}} \approx 0.5 T_{\text{total}}$.

For big systems, time spent in Fourier transformations is small !
Indeed : $T_{FFT} = O(\ln(N_{PW}) N_{PW})$ and $\ln(N_{PW}) \approx 10$ so :

$$N_{proj} N_{at} N_{PW} \gg \ln(N_{PW}) N_{PW}$$

$$T_{\text{nonlop}} \gg T_{FFT}$$

Two ways to implement Conjugate Gradient

More than $> 95\%$ of the computational time is spent in the diagonalization of the Hamiltonian.
At this step : $n(\mathbf{r})$ and H are fixed and we compute new $|\psi\rangle$.

In the conjugate-gradient algorithm, one needs to compute for every band $|\psi\rangle$:

- $\langle\psi|S|\psi\rangle$ and $\epsilon = \langle\psi|H|\psi\rangle$, then $|D\rangle = (H - \epsilon S)|\psi\rangle$
- $\langle D|S|D\rangle$, $\langle D|H|\psi\rangle$, $\langle D|H|D\rangle$ and $\langle\psi'|S|D\rangle$, $\langle\psi'|H|\psi\rangle$, $\langle\psi'|S|\psi\rangle$

Official Abinit version :

$cprj_{ai} = \langle p_{ai}|\psi\rangle$ coefficients (opernla) are computed on the fly and not kept in memory.
$\Rightarrow$ In PAW, need to compute $S|\psi\rangle$ in addition to $H|\psi\rangle$: 2 opernlb operations.
If $nline$ CG steps (typically 4) :

nline+1 opernla and 2nline+2 opernlb per band and per SCF iteration.

Two ways to implement Conjugate Gradient

More than $> 95\%$ of the computational time is spent in the diagonalization of the Hamiltonian.
At this step : $n(\mathbf{r})$ and H are fixed and we compute new $|\psi\rangle$.

In the conjugate-gradient algorithm, one needs to compute for every band $|\psi\rangle$:

- $\langle\psi|S|\psi\rangle$ and $\epsilon = \langle\psi|H|\psi\rangle$, then $|D\rangle = (H - \epsilon S)|\psi\rangle$
- $\langle D|S|D\rangle$, $\langle D|H|\psi\rangle$, $\langle D|H|D\rangle$ and $\langle\psi'|S|D\rangle$, $\langle\psi'|H|\psi\rangle$, $\langle\psi'|S|\psi\rangle$

Official Abinit version :

$cprj_{ai} = \langle p_{ai}|\psi\rangle$ coefficients (opernla) are computed on the fly and not kept in memory.
$\Rightarrow$ In PAW, need to compute $S|\psi\rangle$ in addition to $H|\psi\rangle$: 2 opernlb operations.
If $nline$ CG steps (typically 4) :

nline+1 opernla and 2nline+2 opernlb per band and per SCF iteration.

"cprj_in_memory" version :

$cprj_{ai}$ coefficients (opernla) are computed at the beginning of the computation and stored. They are "propagated" when needed :

$$|\psi^1\rangle = \alpha|\psi^2\rangle + \beta|\psi^3\rangle$$

$$cprj^1_{ai} = \alpha\, cprj^2_{ai} + \beta\, cprj^3_{ai}$$

$\Rightarrow$ One can compute $\langle\psi'|H|\psi\rangle$ or $\langle\psi'|S|\psi\rangle$ without opernlb operations.
No need of $H|\psi\rangle$ or $S|\psi\rangle$! Still one needs $(H - \epsilon S)|\psi\rangle$ so 1 opernlb is needed at every CG step

nline opernla and nline opernlb per band and per SCF iteration.

Promising result with ”cprj_in_memory“ version

```
<BEGIN_TIMER mpi_nprocs = 1, omp_nthreads = 1, mpi_rank = world>
cpu_time =        1552.8, wall_time =        1552.9

routine                          cpu     %       wall     %

getcprj%opernla              318.543  20.5    318.615  20.5
nonlop(apply)                295.517  19.0    295.533  19.0
projbd                       222.625  14.3    222.670  14.3
getcsc%dotprod_g             176.825  11.4    176.862  11.4
vtowfk(ssdiag)               109.697   7.1    109.703   7.1
fourwf%(pot)                  91.825   5.9     91.845   5.9
fourwf%(G->r)                 62.007   4.0     62.021   4.0
subham(dotprod_g)             61.394   4.0     61.400   4.0
getchc%local                  51.159   3.3     51.194   3.3
nonlop%getcsc                 49.992   3.2     50.001   3.2
cgwf_paw%npw_work             19.343   1.2     19.356   1.2
pawcprj(projbd)               18.933   1.2     18.940   1.2
nonlop%getchc                 13.660   0.9     13.660   0.9
```

Development state of new CG routine ($wfoptalg = 10$) :

- works for real WFs, polarized or magnetic systems, and with spin-orbit coupling
- MPI parallelization on k-points (but no paral_kgb) + OpenMP (work in progress)

However :

- almost all tests pass, but not merged in official version yet (work in progress)
- cannot be used for norm-conserving pseudos (not much more efficient anyway)
- not available for Fock exchange, Berry-phase formalism, Nuclear dipole moment

Conclusion

- On 255-Ti : with LOBPCG better performances when increasing the number of threads
- Chebyshev Filtering gives good results too
- Need intensive testing : definition of a “standard set” to measure performances ?

- Non-local operations are the most time-consuming part of the computation for big systems
- “cprj_in_memory” implementation of CG algorithm gives promising results
- What about other algorithms : LOBPCG, Chebyshev ? And paral_kgb ?

Merci de votre attention

Two ways to implement Conjugate Gradient II

First implementation in Abinit :

$$(H\psi)(\mathbf{G}) = K(\mathbf{G})\psi(\mathbf{G}) + \sum_{\mathbf{r}} e^{-i\mathbf{G}\mathbf{r}} V_{loc}(\mathbf{r})\psi(\mathbf{r}) + \sum_{a}^{N_{at}} \sum_{ij}^{N_{proj}} p_{ai}(\mathbf{G}) D_{aij} cprj_{aj}$$

$$(S\psi)(\mathbf{G}) = \psi(\mathbf{G}) + \sum_{a}^{N_{at}} \sum_{ij}^{N_{proj}} p_{ai}(\mathbf{G}) S_{aij} cprj_{aj} \qquad cprj_{ai} = \langle p_{ai}|\psi\rangle$$

$\Rightarrow$ need FFTs for the local part, opernla and opernlb for the non-local part

Then :

$$\epsilon = \sum_{\mathbf{G}} \psi^*(\mathbf{G})(H\psi)(\mathbf{G}) \qquad D(\mathbf{G}) = (H\psi)(\mathbf{G}) - \epsilon(S\psi)(\mathbf{G})$$

"cprj_in_memory" implementation :

If $\psi(\mathbf{G})$, $\psi(\mathbf{r})$ (need FFT) and $cprj_{ai}$ (need opernla) are in memory (same for ψ') :

$\langle\psi'|H|\psi\rangle = \sum_{\mathbf{G}}(\psi'(\mathbf{G}))^* K(\mathbf{G})\psi(\mathbf{G}) + \sum_{\mathbf{r}}(\psi'(\mathbf{r}))^* V_{loc}(\mathbf{r})\psi(\mathbf{r}) + \sum_a \sum_{ij}(cprj'_{ai})^* D_{aij} cprj_{aj}$

$\langle\psi'|S|\psi\rangle = \sum_{\mathbf{G}}(\psi'(\mathbf{G}))^* \psi(\mathbf{G}) + \sum_a \sum_{ij}(cprj'_{ai})^* S_{aij} cprj_{aj}$

$\Rightarrow$ No FFT, no opernla, no opernlb !

$((H-\epsilon S)\psi)(\mathbf{G}) =$

$K(\mathbf{G})\psi(\mathbf{G}) - \epsilon\psi(\mathbf{G}) + \sum_{\mathbf{r}} e^{-i\mathbf{G}\mathbf{r}} V_{loc}(\mathbf{r})\psi(\mathbf{r}) + \sum_a^{N_{at}} \sum_{ij}^{N_{proj}} p_{ai}(\mathbf{G})(D_{aij} - \epsilon S_{aij}) cprj_{aj}$