SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2011-2.3.5 – Second Implementation Phase of the European High Performance Computing (HPC) service PRACE

CAPACITIES

PRACE-2IP

PRACE Second Implementation Project

Grant Agreement Number: RI-283493

# D8.1.2
# Performance Model of Community Codes
*Final*

Version: 1.0
Author(s): Claudio Gheller, Will Sawyer, Thomas Schulthess, CSCS; Fabio Affinito, CINECA; Ivan Girotto, Alastair McKinstry, Filippo Spiga, ICHEC; Laurent Crouzet, CEA; Andy Sunderland, STFC; Giannis Koutsou, Abdou Abdel-Rehim, CASTORC; Fernando Nogueira, Miguel Avillez , UC-LCA; Georg Huhs, José María Cela, and Mohammad Jowkar, BSC.
Date: 24.11.2011

# 5. Performance Analysis of Community Codes: Material Science

The codes discussed in the present chapter are representative set of the most used ab initio electronic structure codes in Europe that are freely distributed under open source licenses. The particular selection is based only on community input. Initial input was gathered during the community selection phase prior to the submission of the first deliverable of this work package. During a subsequent face-to-face meeting as well as several conference calls, representatives from the European Theoretical Spectroscopy Facility (ETSF) and the Quantum ESPRESSO community decided to select the following set of codes that are to be further investigated: (1) a suite of codes from the ETSF; (2) Quantum ESPRESSO [67] including PWscf [66]; and (3) Siesta [68]. Siesta was included as a representative of localized numerical basis set codes that are used mostly in chemistry and complement the plane wave pseudopotential codes developed by the ETSF and Quantum ESPRESSO communities. The ETSF suite of codes includes ABINIT [69], EXCITING [70], ELK [71], OCTOPUS [72], and YAMBO [73].

The code descriptions and performance data presented bellow were provided by the communities that maintain the codes. It has to be emphasized that the benchmarks used for the performance analysis represent typical workloads for which the codes have been developed (in contrast to codes and problems selected by centres to emphasis scalability of a code or performance of a supercomputer). Hence the benchmarks are not uniform. No attempt has been made to polish the results. In some cases, such as ABINITI, teams have in the past invested into scalability of parts of the code base. In other cases, such as EXCITIN/ELK, the code has not been running on much more than large workstations and small clusters. The purpose of subsequent work in the present work package will be to improve the performance of these codes and to map them onto future supercomputing platforms, such as hybrid multi-core systems. The intent is to eventually make these codes fit for PRACE Tier 1 and possibly even Tier 0 systems.

## 5.1 ABINIT

### *5.1.1 Global description of ABINIT*

ABINIT is a package whose main program allows one to find from first principles the total energy, charge density, electronic structure and miscellaneous properties of systems made of electrons and nuclei (molecules and periodic solids) using pseudo-potentials and a plane-wave or wavelet basis. The basic theories implemented in ABINIT are Density Functional Theory (DFT), density-functional perturbation theory (DFPT), Many-Body Perturbation Theory (the GW approximation and Bethe-Salpeter equation), and Time-Dependent Density Functional Theory. The main ABINIT program includes options to optimise the geometry according to the DFT forces and stresses, to perform molecular dynamics simulations using these forces, to determine transition states (string method), to perform path-integral molecular dynamics. It can also directly generate dynamical matrices, Born effective charges, dielectric tensors, and other linear and non-linear coupling quantities, based on Density-Functional Perturbation Theory. Excited states computations from Many-Body Perturbation Theory (the GW approximation) delivers band gaps generally in excellent agreement with experiment, unlike with DFT. Accurate Optical properties are obtained with excitonic effects within the Bethe-Salpether equation.

ABINIT is delivered under the GNU General Public Licence (GPL [74]), and freely distributed on the Web. The documentation is extensive, also provided on the Web. More than 800 automatic tests are integrated in the package, allowing to verify the development by different groups worldwide.

The functional structure of ABINIT, at the highest level, is represented by Figure 40.

Depending on the input parameters, ABINIT will call one (or several) processing unit(s) in turn. These processing units are rather independent, implement different algorithms, and their parallelisation is to be addressed separately.

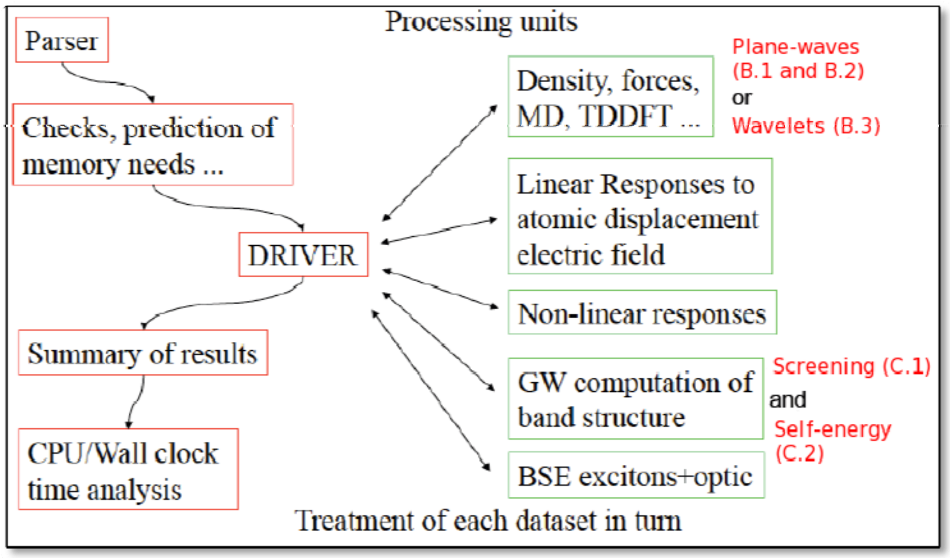

Processing units
Parser
Checks, prediction of memory needs ...
DRIVER
Summary of results
CPU/Wall clock time analysis
Density, forces, MD, TDDFT ...
Plane-waves (B.1 and B.2) or Wavelets (B.3)
Linear Responses to atomic displacement electric field
Non-linear responses
GW computation of band structure
Screening (C.1) and Self-energy (C.2)
BSE excitons+optic
Treatment of each dataset in turn

**Figure 40: Functional structure of ABINIT.**

Section 5.1.2 will address performance issues for the basic DFT calculations using plane-waves and wavelets. In section 5.1.3, the two major steps, based on plane-waves, for the calculation of the electronic structure using MBPT will be detailed: the screening calculation and the self-energy calculation.

In the different sections, several physical or numerical parameters are to be considered in order to understand the computational load and memory scaling, and also the possibilities to distribute it on different processors.

The size of the physical cell (in real space) will usually scale directly with $N_{atom}$, the number of atoms to be represented, although much larger cells will be needed to host systems placed in vacuum (like molecules, or nanotubes, or slabs) when treated with plane-waves.

The basis set size will directly depend on this size of the cell. It will also depend on the spatial resolution, measured by the kinetic energy cut-off $E_{cut}$ (plane-waves) or the grid spacing (wavelets).

The number of points in real space, usually connected to a Fast Fourier treatment (plane-wave case), will be represented by $N_{fft}$. The number of plane-waves $N_{pw}$ to be used is usually a fixed fraction of $N_{fft}$. The number of wavelets will be represented by $N_{wvl}$.

In case of plane-waves, the resolution (small wavelength details) is governed by the kinetic energy cut-off $E_{cut}$. Roughly, $N_{fft}$ or $N_{pw}$ are proportional to $N_{atom}$ times $E_{cut}^{3/2}$. In Ground-state DFT as well as MBPT, different resolution grids might coexists (e.g. for the representation of the screening matrix in GW)

The sampling of electronic velocities (or wave-vectors) is described by the number of "*k*-points" in the Brillouin zone, $N_{kpt}$. Usually it scales inversely proportional to the size of the system, although one cannot go below one *k* point. The number of electronic states, or energy bands $N_{band}$, is also usually proportional to the number of atoms $N_{atom}$. In DFT or DFPT, one often treats explicitly only the occupied electronic states or also the low-lying unoccupied states. For MBPT calculation (screening or self-energy), the number of unoccupied states to be treated to converge the results can be much larger than the number of occupied states (a factor 100 is not unusual).

Finally, there are two additional physical quantities that can lead to a distribution of the computational load and memory:

(1) for spin-polarised systems, the two spins can be treated separately, to a large extend. $N_{sppol}$=2 in this case.

(2) for spinorial wave-functions, the two spinor components can be treated also separately. $N_{spinor}$=2 in this case.

### *5.1.2 Ground-State calculations: performances*

Historically, ABINIT uses plane-waves to describe the electronic wave functions; it makes an intensive use of Fourier transforms, in particular when applying the local part of the Hamiltonian.

ABINIT parallelisation is exclusively performed using the MPI library for the current stable version and for ground-state calculations. In a beta version, several time consuming code sections of the ground-state part have been ported to GPU. Even if it is already useable, this level of parallelisation is in progress…

In recent years, a development of wave functions on a wavelet basis has been introduced (for the ground state calculations), using wavelet transforms and a specific Poisson operator in real space. The implementation of wavelets has been achieved in the project named "BigDFT" [78]. During this project, a library of functions devoted to wavelets has been produced. It is used by ABINIT and can also be called from a standalone executable. The library and the standalone code are inseparable parts of the ABINIT project.

This section devoted to ground-state calculations with ABINIT is divided in three subsections: 1-plane-waves using MPI, 2-plane-waves using CUDA, 3-wavelets.

#### 5.1.2.1 Electronic ground state calculations using planes-waves; performances using MPI

*Parallelisation levels*

Several levels of parallelisation have been introduced in ABINIT; they can be used separately or simultaneously:

- Parallelisation over *k points*: this is a classical level of parallelisation in DFT codes. Several terms of the total energy are obtained by integration over the wave-vector space (*k* points). Each contribution to the integral can be computed separately. A final reduction (global communication) is done to get the total energy. As the scaling of this parallelism level is almost linear, it is not checked here.
- Parallelisation over *independent spins*: in the case of spin-polarised systems, each spin component of the density can be computed independently. As the scaling of this parallelism level is almost linear, it is not checked here.
- Parallelisation over *bands*: parallelisation over *plane-waves*: *These two levels of parallelisation are linked.* To solve the Kohn-Sham DFT equations, an eigenvalue problem has to be solved (only the lowest eigenvalues are computed); $N_{band}$ eigenvalues have to be identified, expressing the Hamiltonian matrix on a plane-wave basis ($N_{pw}$ basis elements). In ABINIT, an iterative "by block" algorithm is used (LOBPCG= "*Locally Optimal Block Preconditioned Conjugate Gradient*"). It can be parallelised over $N_{band}$ and $N_{pw}$. Results concerning this level of parallelisation are presented here.
- Parallelisation over *spinorial components*: in some specific cases (non-collinear magnetism, spin-orbit coupling) each electronic wave function has to be expressed as

a "spinor" (two vectors). This level of parallelisation is implemented in ABINIT but not detailed here.

- Parallelisation over *replicas of the unit cell*: this is a high-level parallelism level. For some specific applications, the simulation cell has to be replicated several times: "*Minimal Energy Path*" research or inclusion of the quantum effect of atomic nuclei (PIMD="*Path-Integral Molecular Dynamics*"). Although it is needed (especially for PIMD), this distribution of workload can be considered as "*embarrassingly parallel*". We just verifired that the scaling is linear.

*Definition of main time consuming parts:*

- *Hamiltonian application*: this routine applies Hamiltonian **H** (and overlap matrix **S**) to the wave-functions. It is divided in three parts:
  - Application of local operator (*Fast Fourier Transform*)
  - Application of non-local operator
  - MPI communications ("*alltoall*")
- *LOBPCG algorithm:* this routine solves the eigenvalue problem by minimisation using *LOBPCG* algorithm; it mainly uses linear algebra.
- *Diagonalisation/orthogonalisation of wave functions:* this routine solves the eigen-problem in wave-functions subspace; it mainly uses linear algebra.
- *Local Potential:* this routine computes the evolving parts of the local potential (*Hartree* + exchange-correlation).
- *Forces:* this routine computes forces on atoms.

*Band-FFT strong scaling*

In the following, we test how the increasing of cores at fixed load (i.e., in a strong scaling regime) affects the performance of each of the functions. We verify that the code scales linealry with the number of cores and try to find at what number of cores a « plateau » is reached. Such tests help to check if the core work is well balanced.

Test case: one vacancy in a 108 Gold atoms unit cell (i.e., 107 atoms). *Gamma k*-point calculation. The (unrelaxed) vacancy breaks symmetries and induces large forces. "Projector Augmented-Wave" (PAW) method is used.

Libraries used: MPI and ScaLAPACK

**Keeping fixed the number of atoms and increasing the number of CPU cores.**

We check here that, mixing band and FFT level, the performance model is not simple. For a given total number of CPU cores, performances directly depend on their distribution on bands and plane-waves.

## Varying only plane-wave CPU cores

| CPU total clock time (s)<br><br>ABINIT part | 65 (band) x 2 (npw)<br><br>*130 cores* | 65 (band) x 4 (npw)<br><br>*260 cores* | 65 (band) x 6 (npw)<br><br>*390 cores* | 65 (band) x 8 (npw)<br><br>*520 cores* |
|---|---|---|---|---|
| Hamiltonian application: local operator (FFT) | 2538,0 | 4547,9 | 8342,9 | 8720,9 |
| Hamiltonian application: non-local operator | 20568,9 | 19451,4 | 18698,3 | 21086,7 |
| Hamiltonian application: *alltoall* communications | 3167,0 | 6613,9 | 12453,1 | 14518,3 |
| LOBPCG algorithm (without Hamiltonian application) | 29385,1 | 74593,0 | 108648,9 | 157520,6 |
| Diagonalisation/orthogonalisation of wave-functions | 1308,0 | 3570,6 | 9277,6 | 15312,9 |
| Local potential computation (*Hartree* + XC) | 8601,4 | 12986,4 | 13081,0 | 15773,6 |
| Forces computation | 3272,8 | 4861,3 | 4918,0 | 5596,9 |
| Others | 6972,4 | 10932,3 | 11858,8 | 21830,1 |
| **Total** | **75813,6** | **137556,8** | **187278,6** | **260360,0** |

**Table 8: CPU total clock time of ABINIT varying the number of plane-wave CPU cores.**

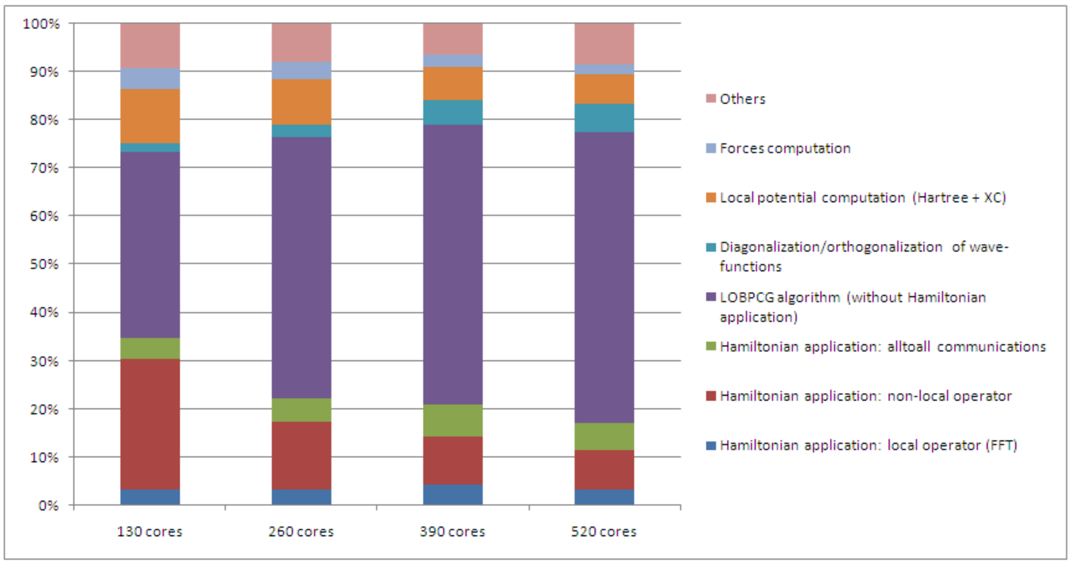


**Figure 41: Repartition of time in ABINIT routines varying the number of plane-wave CPU cores. While some parts of the code scale linearly (ex: non-local operator), others become predominant. A plateau is observed at 390 cores.**

Varying only band cores

| CPU total clock time (s)<br>ABINIT part | 9 (band) x 6 (npw)<br>*72 cores* | 18 (band) x 6 (npw)<br>*144 cores* | 36 (band) x 6 (npw)<br>*216 cores* | 72 (band) x 6 (npw)<br>*432 cores* |
|---|---|---|---|---|
| Hamiltonian application: local operator (FFT) | 4601,9 | 5085,0 | 5196,1 | 6461,3 |
| Hamiltonian application: non-local operator | 20891,1 | 20957,1 | 20982,7 | 23122,3 |
| Hamiltonian application: *alltoall* communications | 3248,7 | 4196,8 | 4708,0 | 10966,4 |
| LOBPCG algorithm (without Hamiltonian application) | 4581,1 | 27232,0 | 50083,6 | 361190,1 |
| Diagonalisation/orthogonalisation of wave-functions | 767,4 | 1731,3 | 2736,7 | 9520,1 |
| Local potential computation (*Hartree* + XC) | 1890,1 | 4063,6 | 6405,4 | 14779,9 |
| Forces computation | 3272,8 | 4861,3 | 4918,0 | 5596,9 |
| Others | -683,7 | 714,2 | 3912,3 | 20955,2 |
| **Total** | 38569,4 | 68841,3 | 98942,8 | 452592,2 |

**Table 9: CPU total clock time of ABINIT varying the number of band CPU cores.**

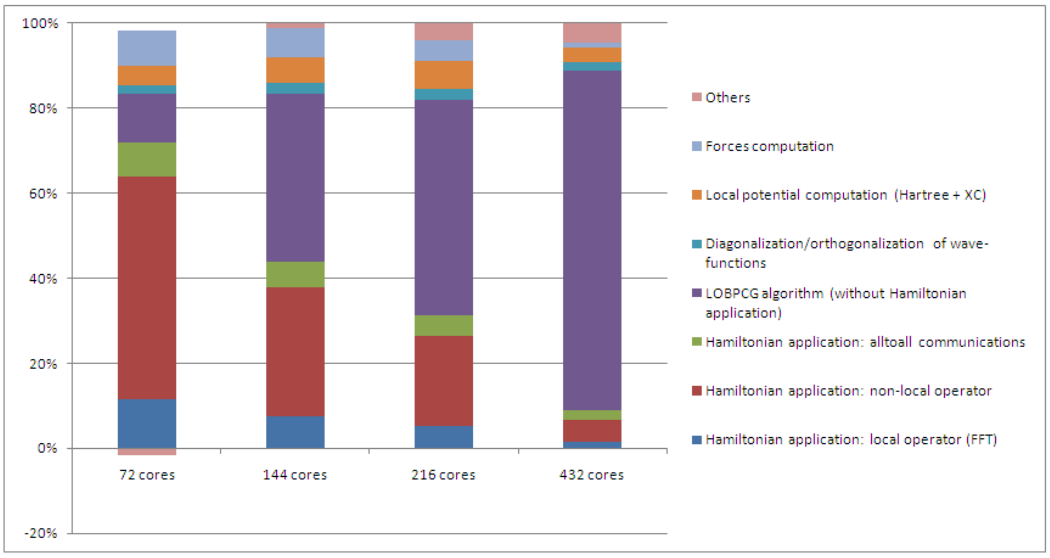


**Figure 42 Repartition of time in ABINIT routines varying the number of band CPU cores. While some parts of the code scale linearly (ex: non-local operator, forces), others become predominant. On 432 cores, the codes clearly has no more a linear behavior.**

*Band-FFT weak scaling*

Differently from the previous two cases presented (strong scaling), we keep here the number of cores fixed observing the performance when the number of atoms changes. In this case the problem size (workload) assigned to each processing element stays constant and additional elements are used to solve a larger total problem.

Test case: a 108 (or 54) Gold atoms unit cell; atomic positions obtained from a Molecular Dynamics simulation at 500K. *Gamma k*-point calculation. The "Projector "Projector Augmented-Wave" (PAW) method is used.

Libraries used: MPI and ScaLAPACK

Varying the number of atoms keeping the number of CPU cores constant

| CPU total clock time (s)<br><br>ABINIT part | 55 (band) x 8 (npw)<br>*108 atoms*<br>*440 cores* | 55 (band) x 8 (npw)<br>*54 atoms*<br>*440 cores* |
|---|---|---|
| Hamiltonian application: local operator (FFT) | 6571,1 | 1852,1 |
| Hamiltonian application: non-local operator | 23775,7 | 1831,9 |
| Hamiltonian application: *alltoall* communications | 8037,5 | 3691,3 |
| LOBPCG algorithm (without Hamiltonian application) | 378221,4 | 32112,2 |
| Diagonalisation/orthogonalisation of wave-functions | 9912,5 | 5485,0 |
| Local potential computation (*Hartree* + XC) | 15262,4 | 3569,5 |
| Forces computation | 5379,0 | 854,5 |
| Others | 21087,2 | 11831,3 |
| **Total** | **468246,8** | **61227,8** |

**Table 10: CPU total clock time of ABINIT varying the number of atoms.**

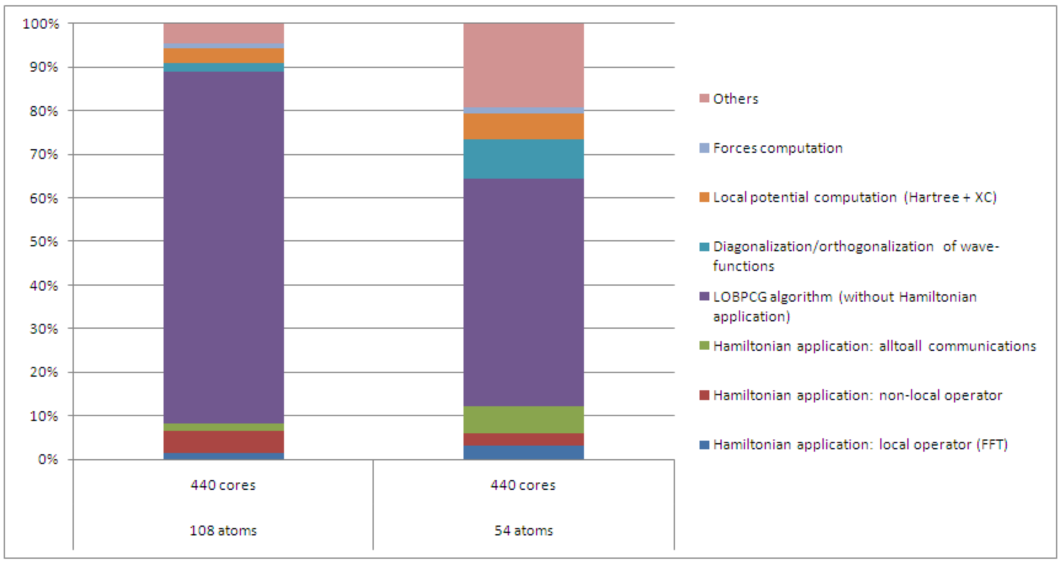


**Figure 43: Repartition of time in ABINIT routines varying the number of atoms. This test case is not a full « weak scaling » performance test as the number of cores is kept fixed. When only the size of the system is increased, the resolution of the eigenvalue problem becomes the predominant part.**

## Varying the number of atoms and the number of CPU cores

| CPU total clock time (s) | 55 (band) x 8 (npw) | 55 (band) x 4 (npw) |
|---|---|---|
| | *108 atoms* | *54 atoms* |
| **ABINIT part** | *440 cores* | *220 cores* |
| Hamiltonian application: local operator (FFT) | 6571,1 | 874,5 |
| Hamiltonian application: non-local operator | 23775,7 | 1759,2 |
| Hamiltonian application: alltoall communications | 8037,5 | 1145,8 |
| LOBPCG algorithm (without Hamiltonian application) | 378221,4 | 14544,8 |
| Diagonalisation/orthogonalisation of wave-functions | 9912,5 | 1234,5 |
| Local potential computation (Hartree + XC) | 15262,4 | 2901,4 |
| Forces computation | 5379,0 | 744,0 |
| Others | 21087,2 | 3370,2 |
| **Total** | **468246,8** | **26574,4** |

**Table 11: CPU total clock time of ABINIT varying the number of atoms and number of cores.**

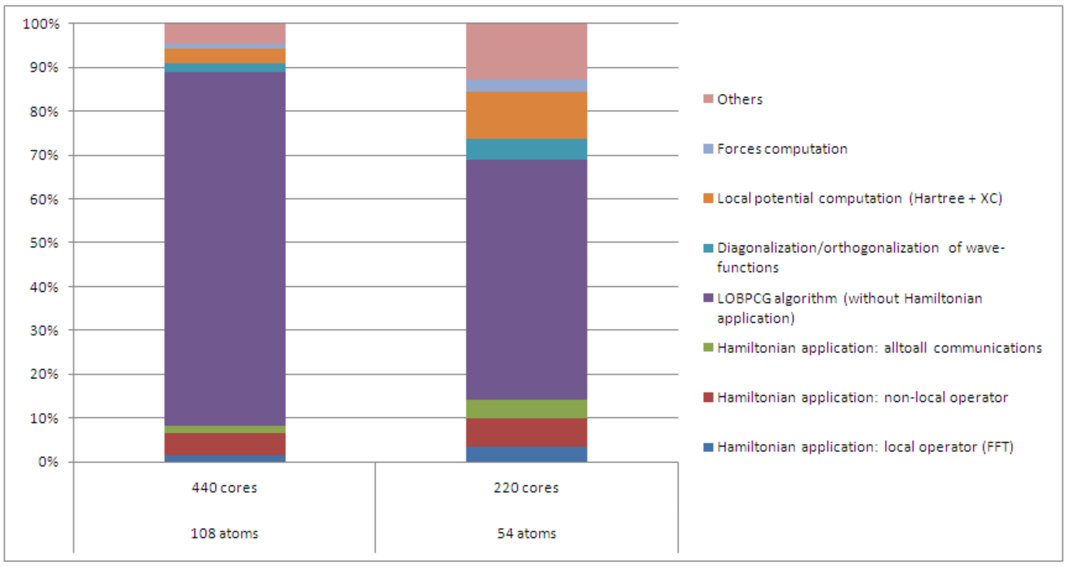


**Figure 44: Repartition of time in ABINIT routines varying the number of atoms and the number of cores. This weak scaling performance test clearly shows that the code does not scale linearly which is an expected behavior for a DFT code. As the size of the simulation cell increases, the number of plane waves increase as the cube of the cell size.**

### *Influence of the CPU cores distribution in the parallelisation levels*

Test case: $PuO_2$ surface: 60 atoms (correlations, magnetism, *f* electrons, vacuum…). 400 bands; *gamma k*-point calculation
"Projector Augmented-Wave" (PAW) method is used.

We check here how the distribution of CPU cores in the ($N_{band}$ x $N_{pw}$ x $N_{kpt}$) levels of parallelisation influences the performance.

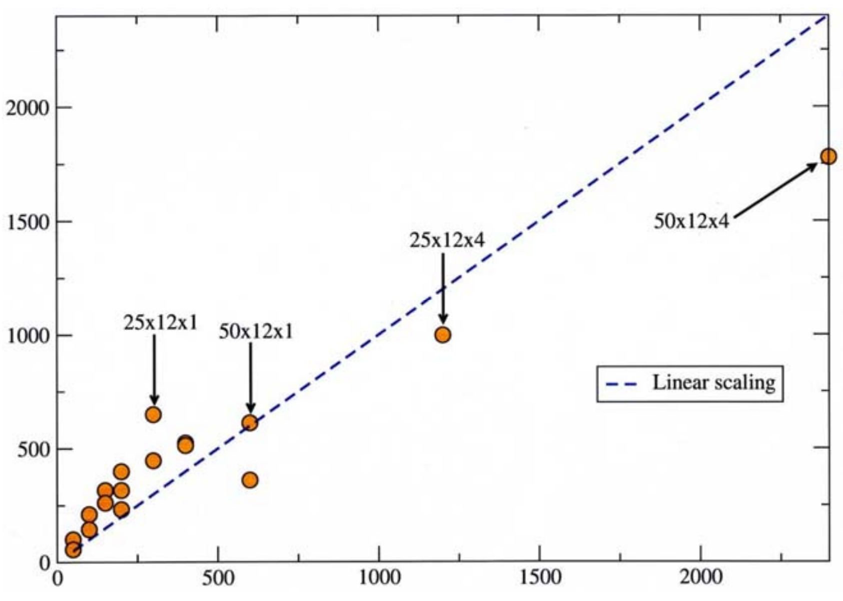
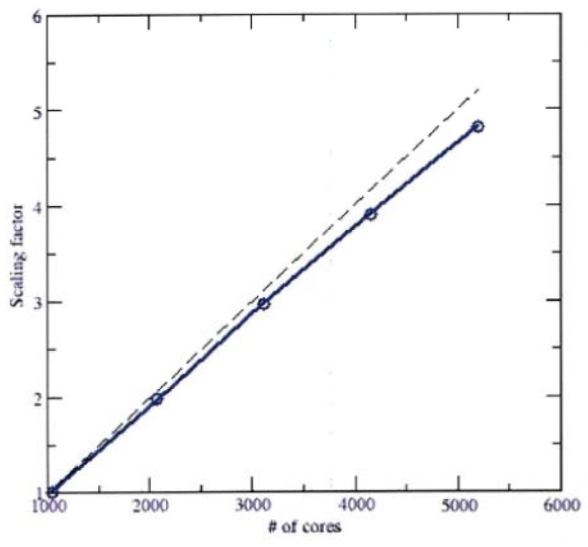


**Figure 45: Scaling of ABINIT wrt the distribution of (Nband x Npw x Nkpt) CPU cores**

*Parallelisation over replicas of the simulation cell*

Test case: calculation of the energy barrier between two positions of a silicon interstitial atom; 65 silicon atoms; 4 *k-points*; 130 bands; PAW method.

We check here that this level of parallelisation scales almost linearly.



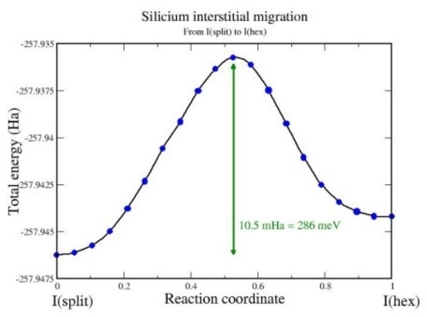


**Figure 46 Scaling of ABINIT wrt the CPU cores distributed on the replicas of the cell.**

### 5.1.2.2 Electronic ground state calculations using planes-waves; performances using CUDA

Use of Graphic Processing Units (GPU) will be available in the 6.12 version ABINIT; this implementation is in beta stage. It uses NVIDIA CUDA library and is still evolving. The following performance profiling has to be considered as a step toward the full implementation.

Three parts of the code have been parallelised:

- Application of the local Hamiltonian (Fast Fourier Transform); it has been chosen to use the *cuFFT* library (included in CUDA package).
- Application of the non-local Hamiltonian; specific CUDA *kernels* have been written for the non-local operator and its derivatives (forces, stresses…).

- Linear algebra used in the LOBPCG algorithm and diagonalisation-othogonalisation in the wave-functions subspace; for that purpose we use *cuBLAS* library delivered in CUDA package and GNU-GPL MAGMA library (LAPACK on GPU [75]).

The GPU implementation is fully compatible with all MPI levels of parallelisation except the FFT level. Only results obtained with one CPU cores are shown here in order to isolate the GPU performances only.

Test cases (all done using PAW):

- ***Test Au***: 107 gold atoms (vacancy in gold crystal)
- ***Test BaTiO$_3$***: 39 atoms (one vacancy in 8 $BaTiO_3$ units)
- ***Test Cu***: 20 copper atoms

*Application of local Hamiltonian*
Use of cuFFT library for the Fast Fourier Transform of wave functions.

| ***Test case*** | ***FFT CPU time (sec)*** | ***FFT GPU time (sec)*** |
|---|---|---|
| ***Test Au*** | 246,1 | 156,2 |
| ***Test BaTiO$_3$*** | 164,2 | 120,3 |
| ***Test Cu*** | 19,6 | 25,4 |

**Table 12: Comparison of elapsed time for the wave function FFT.**

As shown in **Table 12**, executing FFT on GPU is only profitable when the size of wave functions is large enough. However, according to the simulated system, it is possible to send several wave functions together to the graphic card, increasing the FFT efficiency. **Table 13** shows the results obtained for the smallest test case (Test Cu), varying the number of wave functions sent to the GPU:

| ***# of wave-functions sent to the GPU*** | ***FFT GPU time (sec)*** |
|---|---|
| 1 | 25,4 |
| 2 | 20,8 |
| 4 | 14,1 |
| 84 | 9,6 |

**Table 13: Elapsed time for the wave function FFT w.r.t. the number of WF sent.**

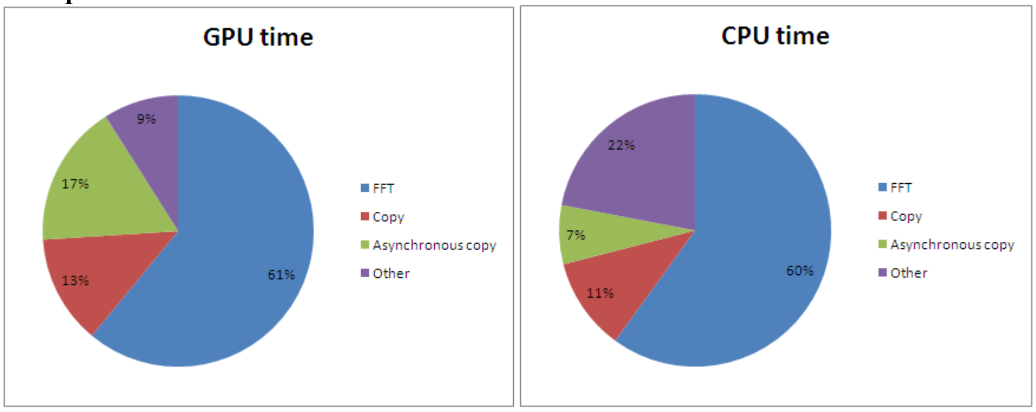


**Figure 47: Profiling of elapsed time for the application of FFT to one wave function, in the "Test Cu" test case; "GPU time" corresponds to the bare GPU time needed by the graphic card to execute the FFT task; "CPU time" corresponds to the total elapsed time, including kernel latencies and synchronisations.**

*Application of non-local Hamiltonian*

Specific CUDA kernels have been implemented to compute application of non-local operator and its contribution to energy, forces and stress tensor.

| ***Test case*** | ***NL op. CPU time (sec)*** | ***NL op. GPU time (sec)*** |
|---|---|---|
| ***Test Au*** | 3142,0 | 1172,9 |
| ***Test $BaTiO_3$*** | 185,1 | 160,2 |
| ***Test Cu*** | 85,1 | 42,4 |

**Table 14: Comparison of elapsed time for the application of non-local operator.**

As shown in **Table 14**, the efficiency of the adopition of the non-local Hamiltonian based approach on the GPU strongly depends on the number and type of treated electrons (*s*, *p*, *d* or *f*).

*Application of linear and matrix algebra*

All vector/matrix multiplication in LOBPCG algorithm are done using the cuBlas library. Orthogonalisation and diagonalisation of the Hamiltonian in the wave-function subspace uses MAGMA package. It can be shown that this use of MAGMA is only profitable when the size of matrixes is large enough. A threshold value has been introduced to call MAGMA routines only when they are efficient. It can be shown that this use of MAGMA is only profitable when the size of matrixes is large enough. A threshold value has been introduced to call MAGMA routines only when they are efficient. This value has been empirically estimated to 100 (size of matrix) and can also be determined « on the fly » by launching a small lapack routine at the start of the code.

| ***Test case*** | ***LOBPCG CPU time (sec)*** | ***LOBPCG GPU time (sec)*** |
|---|---|---|
| ***Test Au*** | 761,9 | 593,1 |
| ***Test $BaTiO_3$*** | 709,0 | 342,1 |
| ***Test Cu*** | 21,3 | 12,3 |

**Table 15: Comparison of elapsed time for the LOBPCG algorithm.**

*Global profiling on GPU*

Performances are subject to change as the development of the GPU code is a work in progress. In the following table we show performances of the ABINIT running on GPU in the present state of the code (v6.12 to be released in december 2011). Improvements of the GPU implementation are ongoing and have not been included.

| ***Test case*** | **Curie GPU Fermi** *Time (sec)* | **Curie CPU** *Time (sec)* | **Titane GPU Tesla** *Time (sec)* | **Titane CPU** *Time (sec)* |
|---|---|---|---|---|
| ***Test Au*** | 609,8 | 1528,0 | 1856,8 | 4230,7 |
| ***Test $BaTiO_3$*** | 681,1 | 865,1 | 810,11 | 849,0 |
| ***Test Cu*** | 67,5 | 235,6 | 102,2 | 153,0 |

**Table 16: Comparison of total elapsed times using (or not) GPU on two different architectures; Curie: CPU=Intel Westmere, GPU=NVidia Fermi M2090; Titane: CPU=Intel Nehalem, GPU=NVidia Tesla S1070**

### 5.1.2.3 Electronic ground state calculations using wavelets: BigDFT profiling

BigDFT is a project that should be considered inseparable from ABINIT. It consists of two (connected) parts: a library, which is used by ABINIT, and a standalone code. We present here some performance of the standalone code.

Two data distribution schemes are used in the parallel version of the program. In the orbital distribution scheme, each processor works on one or a few orbitals for which it holds all its scaling function and wavelet coefficients. In the coefficient distribution scheme each processor holds a certain subset of the coefficients of all the orbitals. Most of the operations — such as applying the Hamiltonian on the orbitals and the preconditioning — are done in the orbital distribution scheme. This has the advantage that we do not have to parallelise these routines with MPI. The calculation of the Lagrange multipliers that enforce the orthogonality constraints onto the gradient as well as the orthogonalisation of the orbitals is done in the coefficient distribution scheme. A global reduction sum is then used to sum the contributions to obtain the correct matrix. Such sums can easily be performed with BLAS-LAPACK routines. Switch back and forth between the orbital distribution scheme and the coefficient distribution scheme is done by the MPI global transposition routine *MPI ALLTOALL(V)*. For parallel computers where the cross sectional bandwidth scales well with the number of processors this global transposition does not require much CPU time. Another time-consuming communication is the global reduction sum required to obtain the total charge distribution from the partial charge distribution of the individual orbital.

#### MPI parallelisation performances. Architecture dependence

The parallelisation scheme of the code is tested since its first version. Since MPI communications do not interfere with calculations, as far as the computational workload is more demanding than time needed for communication, the overall efficiency is always higher than 88%, also for large systems with a large number of processors.

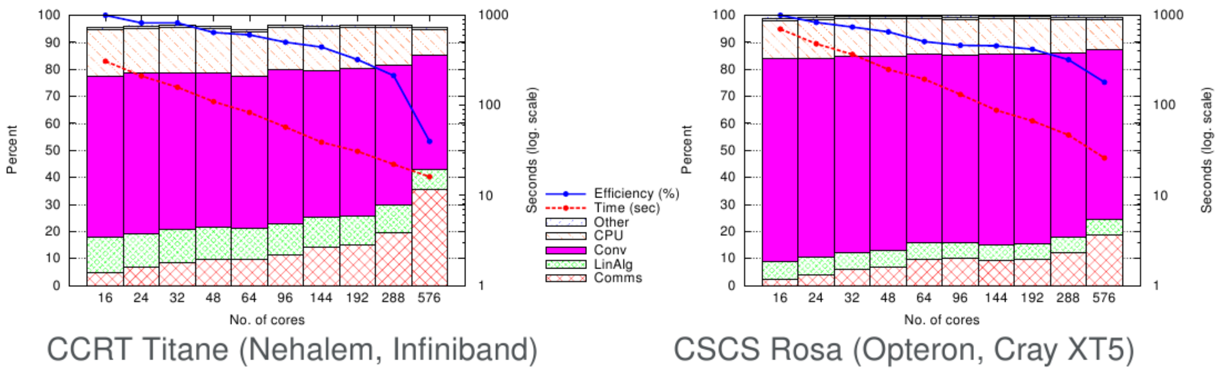


**Figure 48: Comparison of the performances of BigDFT on different platforms.**

Runs on CCRT machine are worse in scalability but better in performances than runs on CSCS one (1.6 to 2.3 times faster). French CCRT Titane platform (Bull Novascale R422 [76]) is compared to Swiss Rosa Cray XT5 [8]. The latter have better performances for communication, and the scalability performances are quite good. However, from the « timeto-solution » viewpoint, the former is about two times faster. This is mainly related to better performances of the linear algebra libraries (Intel MKL [77] compared to Istanbul linear algebra) and of the processor.

#### OpenMP parallelisation

In the parallelisation scheme of the BigDFT code another level of parallelisation was added via *OpenMP* directives. All the convolutions and the linear algebra part can be executed in multi-threaded mode. This adds further flexibility to the parallelisation scheme. Several tests and improvements have been performed to stabilise the behaviour of the code in multilevel

*MPI/OpenMP* parallelisation. At present, optimal performances can be reached by associating one MPI process per CPU, or even one MPI per node, depending on the network and MPI library performances. This has been possible also thanks to recent improvements of the *OpenMP* implementation of the compilers.

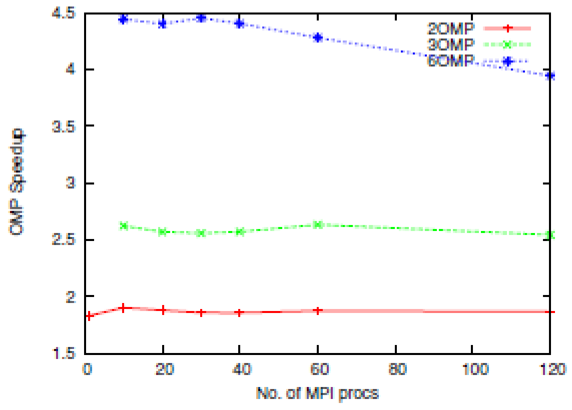


**Figure 49: Speedup of OMP threaded BigDFT code as a function of the number of MPI processes. The test system is a B80 cagem and the machine is Swiss CSCS Palu (Cray XT5, AMD Opteron).**

## GPU acceleration

The BigDFT code is well suited for GPU acceleration. On one hand the computational nature of 3D separable convolutions may allow to write efficient routines, which may benefit of GPU computational power. On the other hand, the parallelisation scheme of BigDFT code is optimal in this sense: GPU can be used without affecting the nature of the communications between the different MPI process. This is in the same spirit of the multi-level *MPI/OpenMP* parallelisation. Porting has been done within the Kronos OpenCL standard, which allows for multi-architecture acceleration.

In the following figure, systems of different sizes have been run in different conditions. The response of the code in the case of an under-dimensioned calculation (where the amount of communication is of the same order as the calculation) has been tested. This may happen if the system is too small, or if the ratio between the runtime GigaFlop/s of the computations and the cross-sectional bandwidth of the network is high.

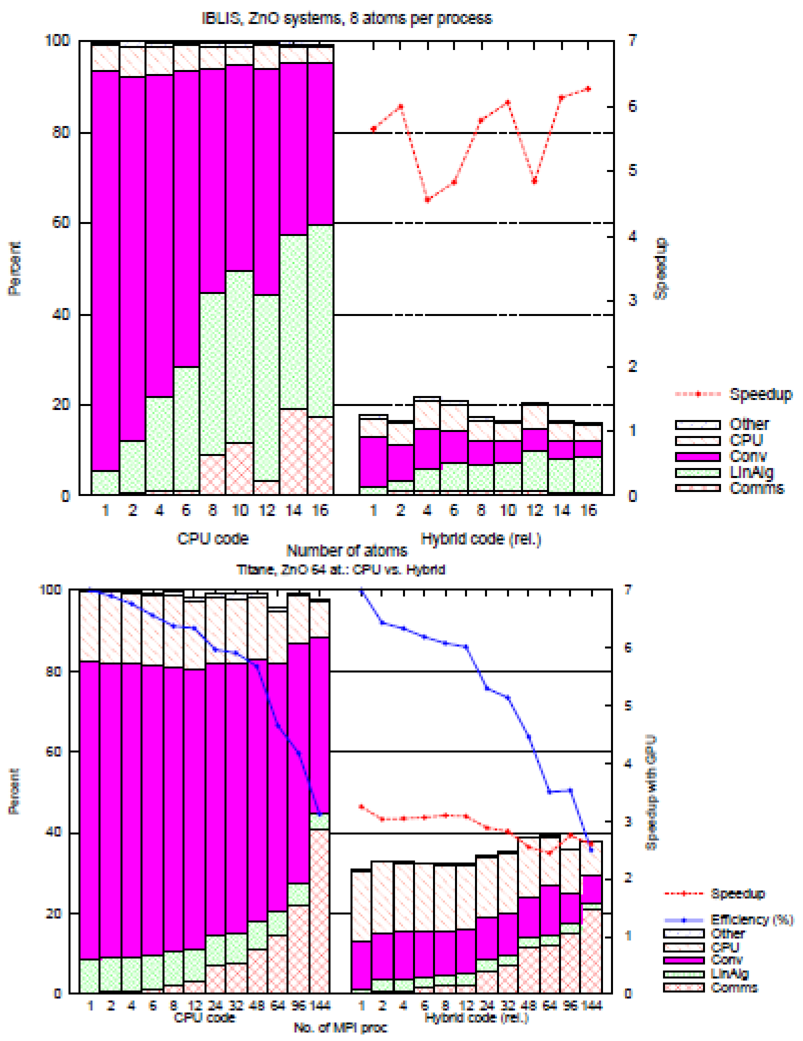


**Figure 50: Relative speedup of the hybrid DFT code wrt the equivalent pure CPU run. In the top panel, different runs for systems of increasing size have been done on a Intel X5472 3GHz (Harpertown) machine. In the bottom panel, a given system has been tested with increasing number of processors on an Intel X5570 2.93GHz (Nehalem) machine. The scaling efficiency of the calculation is also indicated. It presents poor performances due to the fact that the system is too little for so many MPI processes. In the right side of each panel, the same calculation have been done by accelerating the code via one Tesla S1070 card per CPU core used, for both architectures. The speedup is around a value of six for a Harpertown, and around 3.5 for a Nehalem based calculation.**

## *5.1.3 Excited States calculations: performance*

**Description of the test**

The test case is a relaxed 2x2x1 *supercell* of *wurtzite* ZnO with an *oxygen* defect (one oxygen removed). The cell contains 31 atoms, corresponding to 205 occupied electronic bands. The Projector Augmented-Wave (PAW) method is used for all-electron precision, with a 2x2x2 k-point grid in reciprocal space. A *plasmon-pole* model is used for the screening calculation.

For optimal load balancing the number of bands to be calculated was set to 717 and 1229 for the screening calculation, and to 1024 for the self-energy.

### 5.1.3.1 Screening

The most time-consuming parts of the screening calculation are:

*setup*: Initialisation of run

This section reads and checks the header of the KSS file containing the wave functions and electronic band energies and performs the setup of the basic objects needed for computing the screening (pseudo-potentials, PAW objects, GW objects, etc.) This part is not parallelised.

*rdkss*: Reading of the Kohn-Sham orbitals

This routine reads KSS file employing plain Fortran-IO. Each node opens the file and reads a subset of bands. This routine does not scale and it is expected to have a detrimental effect on the scaling.

*qloop*: Matrix inversion and write to file

This routine calculates the inverse dielectric matrix via matrix inversion and writes the SCR file. The inversion is done in serial and the writing is performed by the master node using Fortran-IO primitives. This component does not scale.

*cchi0q0*: Computation of the polarisability for q = 0

*cchi0*: Computation of the polarisability for non-zero q

These routines are parallelised over the empty bands. The implementation scales optimally with the number of processors provided that the number of CPUs divides the number of conduction states used in the calculation.

The tests were executed with ABINIT version 6.5.0.

To decrease the high demand of memory *gwmem*=10 was used, but still at least 16 nodes (8GB memory each) were needed.

**Results**

The functions *cchi0* and *cchi0q0* scale well with the number of processes. This scaling is nearly independent of the number of bands to be calculated, which is different for the total speedup, as to be seen in Figure 51. Apparently the relative cost of the non-scaling functions is higher when calculating fewer bands. Figure 52 compares the partitioning of the workload for different numbers of bands. One can see that the fraction of the well scaling functions (*cchi0* and *cchi0q0*) is 99% when using 16 processes. On 512 processors this decreases to 79% in the 1229 bands case, and even to 64% for 717 bands.

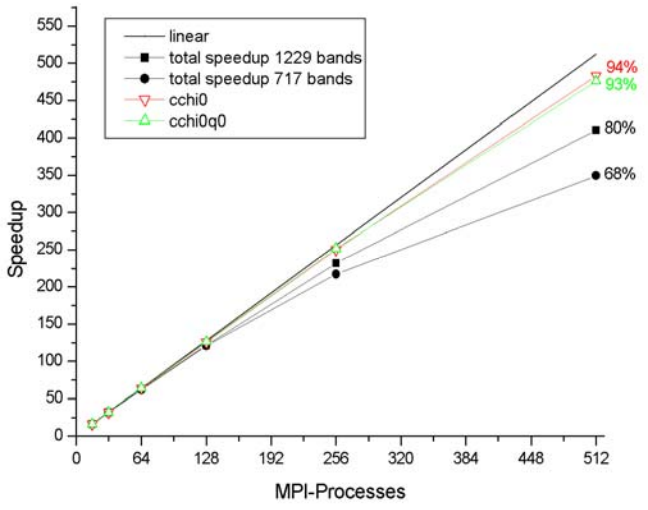
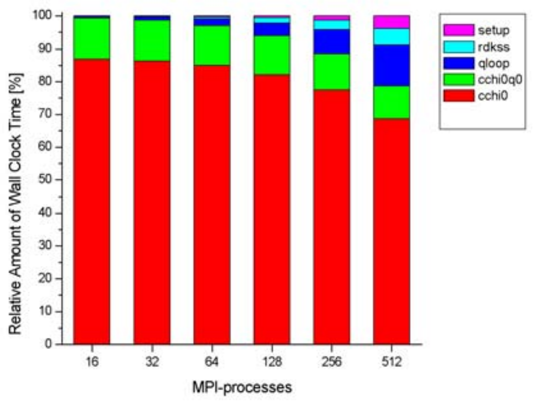


**Figure 51: Speedup for the scaling parts of the screening calculation and total speedup for different numbers of bands**



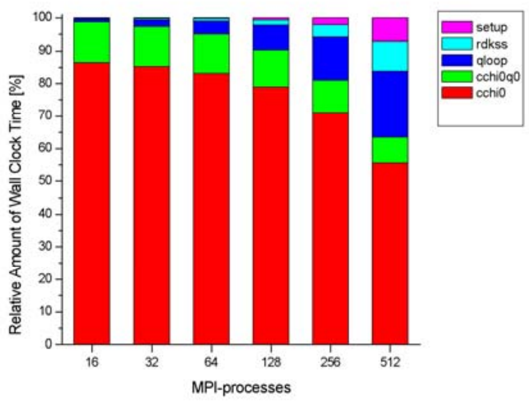


**Figure 52: Relative cost of the most time-consuming code sections On the left for 717 bands, on the right for 1229 bands.**

#### 5.1.3.2 Self-Energy

The calculations in the sigma-part can be decomposed into:

*Init*: Initialisation of the run

*setup*_`sigma`: Initial setup of the self-energy (not parallelised)

*rdkss*: Reading of the Kohn-Sham orbitals (wave function file)

> This routine reads KSS file employing plain Fortran-IO. Each node opens the file and reads a subset of bands. This routine does not scale and it is expected to have a detrimental effect on the scaling.

*csigme* : Calculation of the self-energy matrix elements

> There are two components to be calculated, the exchange contribution and the correlation part. The computation of the correlation (the most time consuming part) should scale up to the total number of bands *occupied+unoccupied* ($N_{band}$). For an optimal distribution the number of processors should divide $N_{band}$.
>
> Note, however, that the calculation of the exchange term will not scale anymore when the number of processor exceeds the number of occupied bands (205 in our case). This should explain part of the degradation of the speedup in csigme when Ncpu >= 128, this limitation can be lifted by implementing a new algorithm that distributes the computation over transitions instead of distributing bands.

Further there is the function *Init2*, whose contribution is negligible and thus will not be discussed here.

The ABINIT version 6.10.1. was used.

For this test the parameter *gwmem*=11 was used, yielding faster but more memory-demanding calculations. As a result it was not possible to run the problem on less than 4 nodes, each featuring 8GB of memory.

**Results**

As to be seen in Figure 53, the only function scaling well is *csigme*, which takes most of the computation time when using a small number of processes (Figure 54). This changes rapidly, decreasing from 99% to 33% when going from 4 to 512 processes. In particular *rdkss* takes over a lot of time.

The resulting total speedup is quite good up to 64 processes, where still an efficiency of 67% can be reached, but decays quickly beyond that point, reaching only 16% at 512 processes.

This result can be expected due to the small number of bands (1024) to be calculated, giving only two bands per processor. Note that *rdkss* (which acts in serial) starts to take up a significant amount of the calculation when the number of processor is comparable to the number of bands.

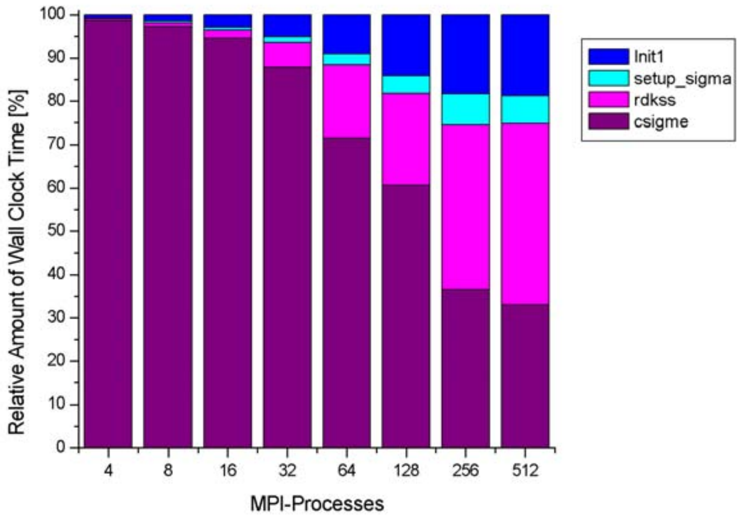


**Figure 53: Speedup for the screening part and its most costly sections**

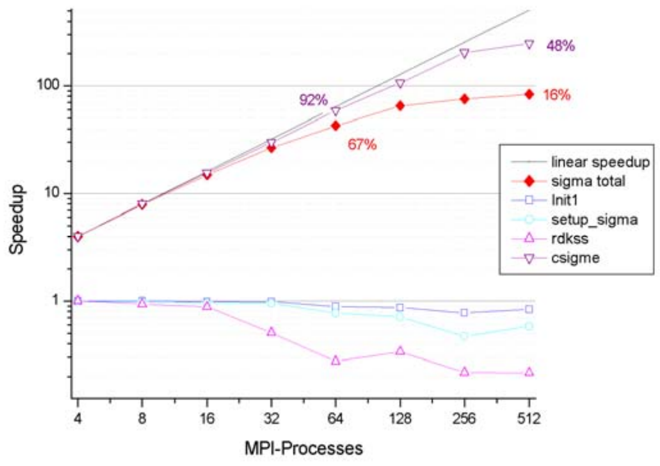


**Figure 54: Relative amount of wall clock time for the partitioning of the sigma calculation.**

## 5.2 Quantum ESPRESSO

### 5.2.1 Description of the code

Quantum ESPRESSO is an integrated suite of computer codes based on density-functional theory, plane waves, and pseudo-potentials - separable, norm-conserving and ultrasoft - and projector-augmented waves. The acronym ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimisation. It is freely available under the terms of the GNU General Public License (GPL). It builds upon newly restructured