SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

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

PRACE-2IP

PRACE Second Implementation Project

Grant Agreement Number: RI-283493

D8.1.3
Prototype Codes Exploring Performance Improvements
Final

Version: 1.0
Author(s): Claudio Gheller and Will Sawyer (CSCS)
Date: 23.12.2011

4 Material Science

4.1 ABINIT

ABINIT [37] is a package, delivered under the GNU General Public Licence (GPL), 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. 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.

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. In recent years, a development of wave functions utilising 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*". 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.

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 work in progress.

As for the performance analysis phase [7], this “performance improvements exploration” phase is divided in three sections: 1-*ground-state calculations using plane waves*, 2-*ground-state calculations using wavelets*, 3-*Excited states calculations*.

4.1.1 Performance improvement: ground-state calculations using plane waves

During the performance analysis phase [7], we identified the critical parts of the code:

- LOBPCG algorithm and diagonalisation/orthogonalisation of wave functions: this routine solves the eigenvalue problem by minimisation using LOBPCG algorithm
- Hamiltonian application: this routine applies Hamiltonian H (and overlap matrix S) to the wave-functions, divided in local operator, non-local operator and communications.
- Forces: this routine computes forces on atoms.

Several other sections have been investigated, but proved to be negligible compared to the previous sections. When looking at the code performances increasing the number of CPU cores, it clearly appears that the main obstacle to scalability is the application of *LOBPCG* algorithm. Only this part will be investigated here.

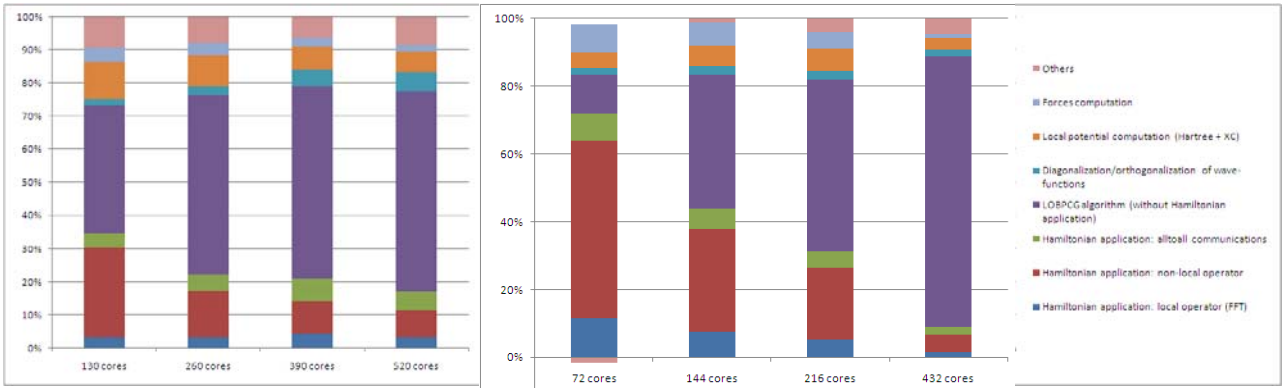

Figure 12: Repartition of time in ABINIT routines: on the left: varying the number of plane-wave CPU cores; on the right: varying the number of band CPU cores.

4.1.1.1 Eigenvalue problem (LOBPCG) improvment: Prototype code improving load balancing; Prototype code using openMP parallelization.

The Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) is an algorithm for finding the smallest eigenvalues and the corresponding eigenvectors of a symmetric positive definite generalized eigenvalue problem.

As shown in the Figure 12, the time spent in the LOBPCG algorithm become predominant; a first (but deep) analysis of the corresponding code section demonstrated that this is essentially due to: 1- an unbalanced load, 2-communications in orthogonalizations/diagonalizations.

In order to improve the performances, both the load balancing must be improved and the communication overhead must be reduced.

The envisaged prototype codes are the following:

1- Unbalanced load correction

The unbalanced load is due to both “band” and “plane wave” distribution. In some disadvantageous cases, some cores can have a load 1.75 times larger than others. Plane wave vectors are incorrectly distributed among processors. Modifying their repartition, according to the physical system, can change this. Only a minor modification at the level of the distribution routine is required, but this has to be done with subtlety. The expected improvement depends on the treated physical system but can reach 50% on some disadvantageous cases.

A prototype code applying the non-local Hamiltonian using a better band/plane wave distribution will be tested.

2- Communications in diagonalizations/orthogonalizations

The analysis shows that the implicated communications mainly are reductions in the orthogonalization and eigenvalue solvers. As there is no hope in decreasing the size of the matrices, the most promising evolution seems to be the use of shared memory parallelism… and *openMP* is the most natural choice. This choice is reinforced when looking at the architecture of some of the *PRACE* nodes. The TGCC-CURIE computer has a “large-nodes” partition with possibly 128 cores on each node.

A prototype code of the orthogonalization routine using *openMP* will be tested.

4.1.1.2 Use of Graphics Processing Units improvement: Prototype code using GPU activation thresholds

Use of Graphic Processing Units (GPU) will be available in the 6.12 version of ABINIT; this implementation is in beta stage. It uses NVIDIA *CUDA* library and is still evolving. Some additional "free of use" packages have been also linked: *NVidia cuBlas*, *NVidia cuFFT*, *MAGMA*.

The GPU implementation is fully compatible with all MPI levels of parallelisation except at the plane wave level. Performance tests show that it is only efficient when the computational load is large enough, i.e. when big physical systems are treated (large simulation cells and/or heavy materials). It is also demonstrated that the efficiency of the GPU code strongly depends on the performance ratio between CPU and GPU.

In order to improve the performance, an envisaged modification is the (automatic) activation of the use of GPU code sections when thresholds are reached. This concerns the *FFT* and the *LaPACK* calls. For these two different parts, the thresholds are necessarily different. This implementation will be based on the automatic determination of thresholds according to the CPU/GPU architecture (only CPU/GPU ratio is important, taking transfers into account). This could be made by calling small FFT or *LaPACK* routines at start simultaneously on CPU and GPU and comparing the obtained performances. Then, according to the physical system size, the code could decide to use CPU or GPU version of the algorithms.

A prototype code using an automatic process to determine whether GPU has to be used for *FFT* or *LaPACK* will be tested.

4.1.2 Performance improvement: ground-state calculations using wavelets (BigDFT)

BigDFT uses three level of optimization:

- *MPI* over orbitals for coarse parallelization,
- *OpenMP* for each orbital for fine optimization,
- *OpenCL* to accelerate the code by means of GPUs.

As in the plane wave version of ABINIT, we would first improve the *OpenMP* optimization of *BigDFT* especially for the exchange-correlation potential calculation (*LibXC*). This task is straightforward and should give some significant performance imporvements for small simulated systems. This will represent a gain between 10 and 20% depending on the number of threads.

A prototype code calculating the exchange-correlation potential with openMP directives will be tested.

The second task would be the improvement of the Poisson solver used both by *BigDFT* and ABINIT. The Poisson solver is based on *FFT* (Fast Fourier Transform) in order to calculate long range solution of the Poisson equation. The second task would be the use of *OpenCL* to calculate *FFT* on GPUs in order to accelerate the Poisson solver. This will represent a small gain for large systems but a substantial gain for small systems. The main interest is to have an *OpenCL* version of *FFT* is to accelerate also the exchange-correlation calculations in the specific case of *hybrid functionals*. In this case, the gain will be substantial for large systems. The calculation of the exchange-correlation for hybrid functionals represents 90% of the time. Using GPUs, we will reduce the execution time by a factor of 6. This will give a gain of 4 for the whole code.

We point out that this speedup could be also realised by means of *OpenMP* directives. The advantage of using GPU is important if we can use both *OpenMP* and GPU for different operations in the code. This will be realised in future developments.

This *OpenCL* FFT could be also used in future developments for the excited states calculation which use a large number of FFTs.

A prototype code will be tested with *OpenCL FFT* for all long range convolutions.

4.1.3 Performance improvement: excited states calculations

GW calculations are very CPU and memory demanding due to the large number of empty bands that are usually needed to converge the *quasiparticle* corrections. For this reason, the *GW* code of ABINIT distributes the full set of bands at the beginning of the run such that each Processing Unit can compute locally its own partial contribution without having to exchange data with the other MPI nodes involved in the run. Most of the computation is distributed among the node and a few collective MPI communications are required to gather the final results.

The preliminary tests performed at the Barcelona Supercomputing Center have revealed that both the computation of the polarizability and of the *self-energy* matrix elements scale well up to 512 processors. The degradation of the total speedup observed for large number of processors is mainly due to:

- The reading of the orbitals from the KSS file
- The matrix inversion performed during the computation of the screening.
- An unbalanced distribution of the work during the calculation of the exchange part of the self-energy.

The envisaged prototype codes are the following:

1- Reading of the orbitals

In order to achieve better scaling in the I/O part, we plan to replace the plain Fortran-IO implementation with a new version based on MPI-IO.

A prototype code using collective MPI-IO routines to read the orbitals will be tested.

2- Matrix inversion

The matrix inversion represents a serious bottleneck for large-scale applications since the CPU time quickly increases with the number of atoms.

A prototype code using *ScalaPACK* routines to to perform the inversion will be tested.

3- Unbalanced distribution

The unbalanced distribution of the work in the calculation of the exchange part of the self-energy can be avoided by resorting to a different distribution of the orbitals such that each node can participate in the computation.

A prototype code employing a different MPI distribution of the orbitals will be implemented and tested.

Another point worth stressing is that the present implementation is best suited for crystalline compounds where symmetries can be used to reduce the number of independent matrix elements that have to be computed explicitly. Isolated or disordered systems can be treated by means of the supercell technique but, in this case, the implementation is not optimal as the

non-scalable arrays whose size increases with the number of atoms will dominate the memory requirements.

To overcome this limitation we plan to use a hybrid MPI-OpenMP implementation in which MPI is used at a coarse level in order to distribute the most memory-demanding arrays while *OpenMP* is employed for the fine grain parallelization.

A prototype code employing *OpenMP* to parallelize the most CPU intensive parts (FFT, matrix algebra, loops,…) will be implemented and tested.

4.2 Quantum ESPRESSO

Quantum ESPRESSO [38] is a distribution of software for atomistic simulations based on electronic structure using density-functional theory with a plane-wave basis set and pseudo potential. Main packages are PWscf (PW), a self-consistent electronic structure solver, and CP, a variable-cell Car Parrinello molecular dynamics package.

The flowchart of the PW code is shown in the figure below. This is the typical flowchart of a density-functional calculation: in the self-consistency cycle, once the wave functions are calculated from the input, the diagonalization of the Hamiltonian matrix is performed for each k-point. As we showed in D8.1.2, this is certainly the most expensive part of the self-consistency cycle. Then the charge density is calculated and from that new potentials are generated.

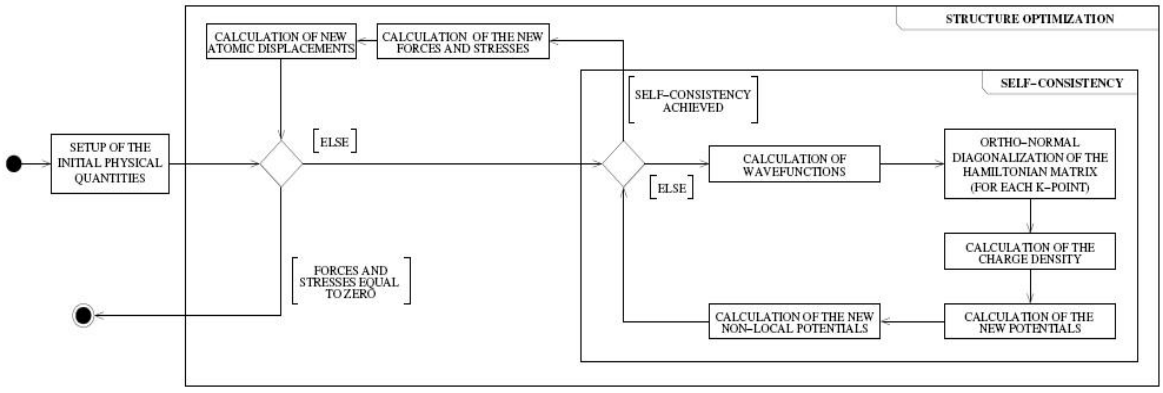

Figure 13: Schematic UML activity diagram of PWscf code.

The computational cost of the diagonalization of the hamiltonian matrix is strictly related to the physical properties of the system under investigation. PW implements two different methods to achieve this task: a Davidson method and the Conjugate Gradient method. Within this project we aim to focus on the Davidson method since it is widely used also in other Quantum chemistry codes.