Available on-line at www.prace-ri.eu

**Partnership for Advanced Computing in Europe**

# Improvements of BigDFT code in modern HPC architectures

Luigi Genovese[a,b,*], Brice Videau[a], Thierry Deutsch[a], Huan Tran[c], Stefan Goedecker[c]

[a]Laboratoire de Simulation Atomistique, SP2M/INAC/CEA, 17 Av. des Martyrs, 38054 Grenoble, France
[b]European Synchrotron Radiation Facility, 6 rue Horowitz, BP 220, 38043 Grenoble, France
[c]Institut für Physik, Universität Basel, Klingelbergstr.82, 4056 Basel, Switzerland

**Abstract**

Electronic structure calculations (DFT codes) are certainly among the disciplines for which an increasing of the computational power correspond to an advancement in the scientific results. In this report, we present the ongoing advancements of DFT code that can run on massively parallel, hybrid and heterogeneous CPU-GPU clusters. This DFT code, named BigDFT, is delivered within the GNU-GPL license either in a stand-alone version or integrated in the ABINIT software package. Hybrid BigDFT routines were initially ported with NVidia's CUDA language, and recently more functionalities have been added with new routines writeen within Kronos' OpenCL standard. The formalism of this code is based on Daubechies wavelets, which is a systematic real-space based basis set. The properties of this basis set are well suited for an extension on a GPU-accelerated environment. In addition to focusing on the performances of the MPI and OpenMP parallelisation the BigDFT code, this presentation also relies of the usage of the GPU resources in a complex code with different kinds of operations. A discussion on the interest of present and expected performances of Hybrid architectures computation in the framework of electronic structure calculations is also adressed.

Project ID:

## 1. Introduction and context

In 2005, the EU FP6-STREP-NEST BigDFT project funded a consortium of four European laboratories (L_Sim - CEA Grenoble, Basel University - Switzerland, Louvain-la-Neuve University - Belgium and Kiel University - Germany), with the aim of developing a novel approach for DFT calculations based on Daubechies wavelets. Rather than simply building a DFT code from scratch, the objective of this three-years project was to test the potential benefit of a new formalism in the context of electronic structure calculations.

As a matter of fact, Daubechies wavelets exhibit a set of properties which make them ideal for a precise and optimized DFT approach. In particular, their systematicity allows to provide a reliable basis set for high-precision results, whereas their locality (both in real and reciprocal space) is highly desired to improve the efficiency and the flexibility of the treatment. Indeed, a localized basis set allows to optimize the number of degrees of freedom for a required accuracy (see [4]), which is highly desirable given the complexity and inhomogeneity of the systems under investigation nowadays. Moreover, an approach based on localized functions makes possible to control explicitly the nature of the boundaries of the simulation domain, which allows to consider complex environments like mixed boundary conditions and/or systems with a net charge.

The wavelet properties are also of great interest for an efficient computational implementation of the formalism. Thanks to wavelets formalism, in BigDFT code, the great majority of the operations are convolutions with short and separable filters. The experience of the BigDFT team in optimizing computationally intensive programs made BigDFT a computer code oriented for High Performance Computing. Since late 2008, BigDFT is able to take advantage of the power of the new hybrid supercomputers, based on Graphic Processing Units (GPU). So far, this is the sole DFT code based of systematic basis sets which can use this technology. In 2009, L. Genovese has received the French Bull-Fourier prize for having implemented and coordinated the realization of the hybrid version of BigDFT.

## 2. BigDFT basic operations: 3D convolutions

In the Kohn-Sham (KS) formulation of DFT, the electrons are associated to wavefunctions (orbitals), which are represented in the computer as arrays. In wavelets formalism, the operators are written via convolutions with

*Corresponding author.
tel. +0-000-000-0000 fax. +0-000-000-0000 e-mail. luigi.genovese@cea.fr

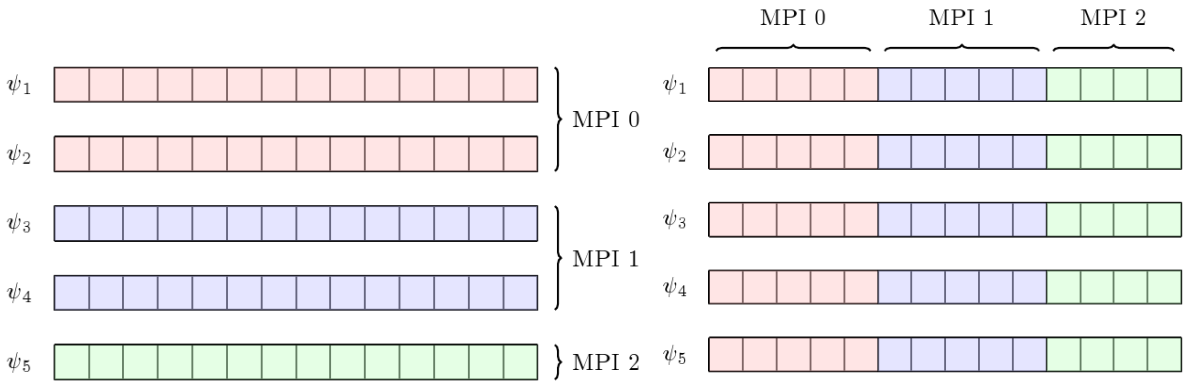



Fig. 1. Orbital(left) and coefficient (right) distribution schemes.

short, separable filters. The detailed description of how these operations are defined is beyond the scope of this report and can be found in the BigDFT reference paper [4].

Convolutions are among the basic processing blocks of BigDFT. Special care has to be taken regarding their performances. The CPU convolutions of BigDFT have thus been thoroughly optimized. The convolutions can be expressed with three nested loops. A three-dimensional array $s$ (input array) of dimension $n_1, n_2, n_3$ (the dimension of the simulation domain of BigDFT) is transformed into the output array $\Psi_r$ given by

$$\Psi_r(I_1, I_2, I_3) = \sum_{j_1, j_2, j_3 = -L}^{U} \omega_{j_1} \omega_{j_2} \omega_{j_3} s(I_1 - j_1, I_2 - j_2, I_3 - j_3) \; . \tag{1}$$

With a lowercase index $i_p$ we indicate the elements of the input array, while with a capital index $I_p$ we indicate the indices after application of the filters $\omega_i$, which have extension from $-L$ to $U$. The most convenient way to calculate a three dimensional convolution of this kind is by combining one dimensional convolutions and array transpositions, as explained in [11, 5, 6]. Despite their simplicity, optimizing convolutions is a far-from-trivial task. The unusual memory access patterns combined with the transposition of the result defeat the compiler optimizer. For instance, an unoptimized Magic Filter convolution of BigDFT on a *Intel(R) Xeon(R) CPU X5550 @ 2.67GHz* yields 0.55 GFLOPS which barely represents 5% of peak performances (10.7 GFLOPS).

A first optimization pass has been applied on the Fortran code directly. Unrolling the outer loop allows reuse of the filter value and index calculations for outer elements. Using standard optimization, tis code runs at 2.4 GFLOPS, which represents 22% of peak performances. But this code can also be compiled using automatic vectorization and reaches 3.4 GFLOPS corresponding to 32% of peak performances.

Fortran optimizations granted an acceleration of a factor 6. Nonetheless, this result is unsatisfactory regarding peak attainable performances. The code was thus hand-tuned in order to make better use of the available vector engine. Several dozens of memory access patterns and vectorizing schemes were designed and tested in order to find the most efficient. The resulting Magic Filter convolution runs at 7.7 GFLOPS yielding 72% of peak performances. This convolution is 13 times faster than the unoptimized one.

This optimization process is difficult and tedious, but the regularity in tested patterns may allow automated pattern generation and algorithm tuning. If this proves successful every convolution of BigDFT will be optimized accordingly in the future.

## 3. Parallelisation

Two data distribution schemes are used in the parallel version of our 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 (see Fig.1) 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 is done in the orbital distribution scheme. This has the advantage that we do not have to parallelize these routines with MPI. The calculation of the Lagrange multipliers that enforce the orthogonality constraints onto the gradient as well as the orthogonalization of the orbitals is done in the coefficient distribution scheme (Fig. 1). A global reduction sum is then used to sum the contributions to obtain the correct matrix. Such sums can esily be performed with the very well optimized BLAS-LAPACK libraries. 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 [7] scales well with the number of processors this global transposition does not require a lot of 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.

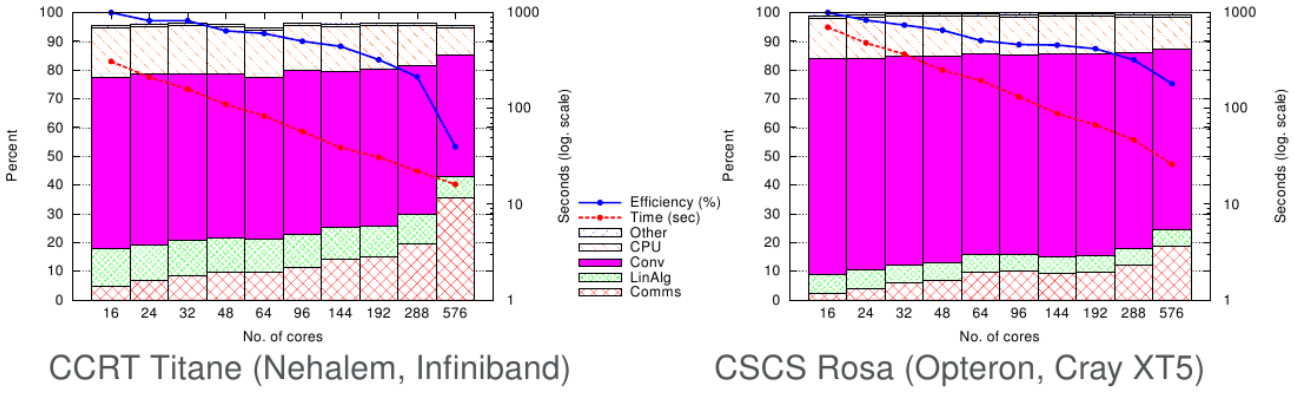



Fig. 2. Comparison of the performances of BigDFT on different platforms. Runs on CCRT machine are worse in scalability but better in performances than runs on CSCS one (1.6 to 2.3 times faster).

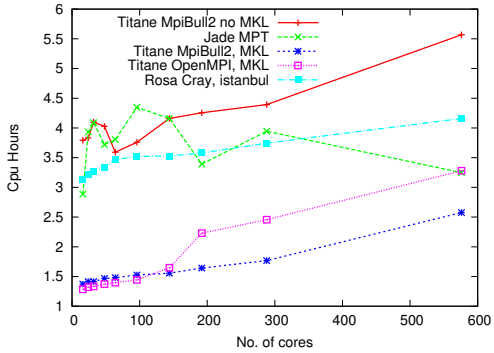



Fig. 3. Number of CPU hours needed to terminate the job of Fig. 2 in different architectures with different MPI and linear algebra libraries.

### *3.1. MPI parallelisation performances. Architecture dependence*

The parallellisation 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 need to communication, the overall efficiency is always higher than 88%, also for large systems with a big number of processors, see Ref. [4].

We have evaluated the amount of time spent for a given operation on a typical run. To do this we have profiled the different sections of the BigDFT code for a parallel calculation. In Fig.2 we show the percent of time which is dedicated to any of the above described operation, for runs with two different architectures: French CCRT Titane platform (Bull Novascale R422) is compared to Swiss Rosa Cray XT5. The latter have better performances for communication, and the scalability performances are quite good. However, from the "time-to-solution"'viewpoint, the former is about two times faster. This is mainly related to better performances of the linear algebra libraries (Intel MKL compared to Istanbul linear algebra) and of the processor. These benchmarks are taken for a run of the BigDFT code with the same input files (a relatively small Benchmark system of 128 atoms of ZnO), starting form the same sources. Then we have performed the same experience, with the same system) different machines, to see how the relative performances between the communications and the libraries may influence end-user behaviour of the code. Results are presented in Fig. 3. It can be seen that overall results may be significantly affected by these parameters. It is worth noticing that, in this case, Fig. 2 shows that parallel efficiency is not always a significative evaluation parameter.

### *3.2. OpenMP parallelisation*

In the parallelisation scheme of the BigDFT code another level of parallelisation was added via OpenMP directive. In particular, all the convolutions and the linear algebra part can be executed in multi-threaded mode. This add further flexibility on the parallelisation scheme. Several tests and improvements have been performed to stabilise the behaviour of the code in multilevel MPI/OpenMP parallelization. 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. Utilities to profile the code behaviour in are under implementation at the moment, such as to identify the possible bottlenecks at runtime. In Fig. 4 the efficiency of the OMP

parallelisation is presented for the full code in another test case (a $B_{80}$ system). It is important to show that this is weakly affected by the number of MPI processes.

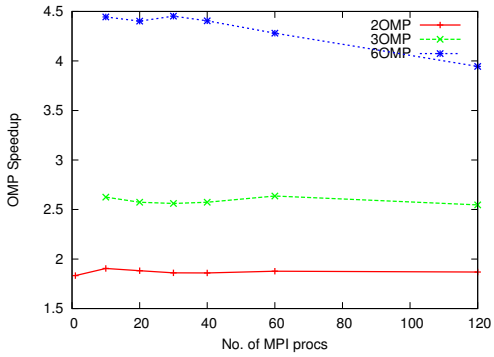



Fig. 4. Speedup of OMP threaded BigDFT code as a function of the number of MPI processes. The test system is a $B_{80}$ cagem and the machine is Swiss CSCS Palu (Cray XT5, AMD Opteron).

## 4. GPU acceleration

As it was presented in [5, 6], the operation of the BigDFT code are well suited for GPU acceleration. Indeed, 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 parallilisation. Porting has been done within Kronos' OpenCL standard, which allows for multi-architecture acceleration. An example of this will be given in the following.

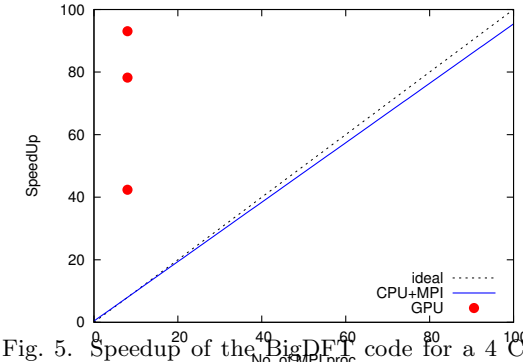
Without entering into the details of GPU porting of BigDFT, let us examine the overall performances of the code. In Table 5, we have presented an optimal case for GPU acceleration. For this run, most of the operations can be GPU accelerated. The overall speedup of the full code is of about one orded of magnitude. This is really encouraging for challenging performances on bigger hybrid machines.

In Figure 6, systems of different sizes have been run in differnent conditions. In particular, what have been tested is the response of the code in the case of an under-dimensioned calculation, where the amount of communication is of the same order of the calculation. This may happen if the system is too little, or if the ratio between the runtime GFlops of the computations and the cross-sectional bandwidth of the newtwork is high.

In any case, the code appears to have a robust behaviour even in non-optimal conditions for the GPU acceleration. Even in this case, an improvement of the time-to-solution of a factor of around 3 has to be expected. This is interesting since for a basic usage the end-user is not required to understand deeply the optimal dimensioning of the system for the architecture.

### *4.1. Hybrid* and *Heterogeneous architectures*

The OpenCL standard allows for multi-platform accelerations. This means that a BigDFT executable is able to be executed in parallel evnironment where the machines have different accelerators. As an example, we have




| # GPU added | 2 | 4 | 8 |
|---|---|---|---|
| SpeedUp (SU) | 5.3 | 9.8 | 11.6 |
| # MPI equiv. | 44 | 80 | 96 |
| Acceler. Eff. | 1 | .94 | .56 |

Fig. 5. Speedup of the BigDFT code for a 4 Carbon atoms supercell (Graphene), with 113 K-points. Calculation is performed with 8 MPI processes on the CEA-DAM INTI machine, based on Westmere processors and NVidia Fermi. For each run, the number of equivalent MPI processes is indicated, given that the parallel efficiency of this run is of 98%. Also the efficiency of the GPU acceleration is presented.

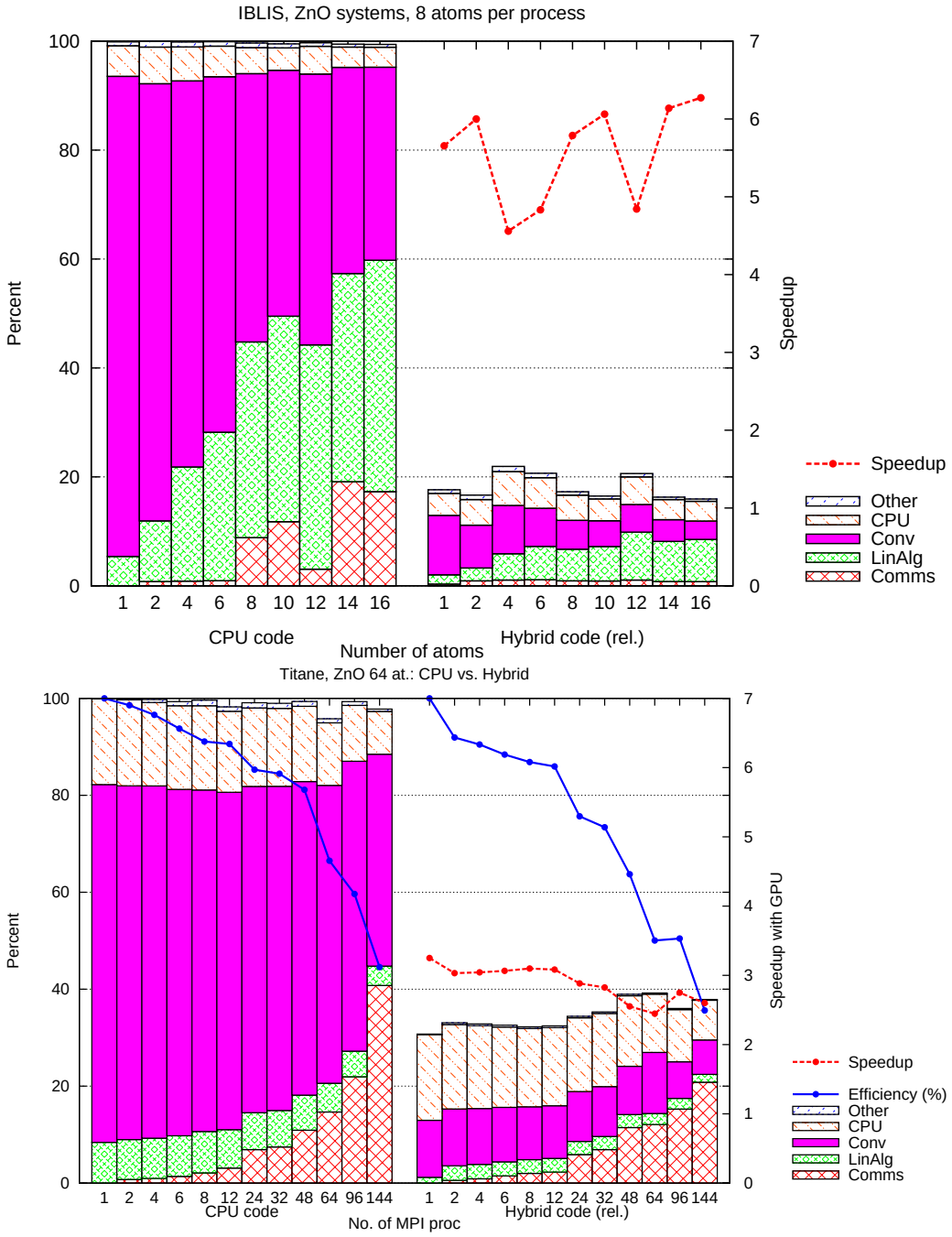



Fig. 6. Relative speedup of the hybrid DFT code wrt the equivalent pure CPU run. In the top panel, different runs for sustems of increasing size have been done on a Intel X5472 3GHz (Harpertown) machine. In the bottom panel, a given system have been tested with increasing number of processors on a Intel 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.

| MPI + Nvidia(NV)/ATI | 1 | 1 + NV | 4 + NV | 1 + ATI | 4 + ATI | (4 + NV) + (4 + ATI) |
|---|---|---|---|---|---|---|
| Execution Time (s) | 6020 | 300 | 160 | 347 | 197 | 109 |
| Speedup | 1 | 20.07 | 37.62 | 17.35 | 30.55 | 55.23 |

Table 1. Parallel run of BigDFT on a hybrid Etherogeneous architecture. Two QuadCore Nehalem nodes are connected to two different cards, one ATI and one NVidia. The system is a 4 Carbon atom surface supercell with 52 k-points.

| MPI*OMP + GPU | 1*1 | 32*1 | 64*1 | 128*1 | 32*6 | 128*6 | 128*1 + 128 |
|---|---|---|---|---|---|---|---|
| SCF iteration (s) | 170 | 7.2 | 3.8 | 2.0 | 1.5 | .44 | .30 |
| Force evaluation (s) | 2210 | 93.6 | 49.4 | 26.0 | 19.5 | 5.72 | 3.92 |
| AIMD steps/day | 40 | 923 | 1749 | 3323 | 4430 | 15104 | 22040 |
| MD ps/day | 0.02 | 0.461 | 0.874 | 1.661 | 2.215 | 7.552 | 11.02 |

Table 2. Effect of the MPI parallelisation OMP, and GPU acceleration on the complete molecular dynamics of a system of 32 $H_2O$ molecules.

executed a test run on two Nehalem Quad-Core machines with one GPU each, one NVidia Fermi and one ATI Radeon HD 2600 XT. Results can be found in Table 1 The code runs as expected. In other terms, the MPI efficiency is added to the acelerated code performances, accordingly to the peculiar behaviour of the architecture. Clearly, different architectures have different features and a kernel optimisation should be performed differently. However, since OpenCL kernels are compiled at runtime, some typical parameters can be extracted before generating the OpenCL binaries. For the time being, the NVidia optimised kernels are used in BigDFT, even for the ATI runs. However, some parameters like the maximum number of threads per kernel execution (NVidia blocks) are extracted at the moment of compilation. The run thus becomes really etherogeneous, i.e. each of the machine has at the end a different binary code.

At present only the convolutions are accelerated, but strategies are under evaluation for accelerate also other operations of BigDFT code, like the FFT, which is of great importance for advanced features like the evaluation of the exact exchange operator. Strategies of automatic code generation at runtime are under investigation.

## 5. End-user benefits and Future work

BigDFT code is routinely used for production runs on different case studies, both in Physics and Chemistry. In Table 2 we present the advantages of accelerating BigDFT code in a multi-level parallelisation framework, by giving the number of steps which are accessible in one day for a Ab Initio Molecular Dynamics simulation. Both OpenMP and GPU parallelisation level allows an improvement of the data which can be collected.

Wavelets are also interesting for DFT due to their compatibility with Order N methods, and a O(N) version of the BigDFT code is at present under preparation. Thanks to BigDFT software engineering, this version will be able to benefit from the existing multi-level parallelisation layers. To this aim, the efficiency of different communication schemes of the BigDFT code will be investigated, in particular with respect to the multi-level parallelisation features. Simple collective, blocking communication schemes will be tested against more elaborated non-blocking, point-to-point parallelisation algorithms, both with traditional and O(N)-like data repartitions. These investigations should help in finding the correct dimensioning of the approach with respect to the particular system under investigation. This fact should improve the know-how on parallel DFT approaches, such as to improve performances of other similar codes.

In the following studies, the performance of the communications of BigDFT as a function of the number of MPI process will be the principal quantity. Indeed, in most of the architectures the MPI_ALLTOALLV approach has revealed to be a bottleneck for systems with more than 1000 MPI processes. The main idea of this project is to understand how to reduce the overhead related to communications, by changing scheme and/or by a clever usage of multi-level MPI/OpenMP paradigms.

As anticipated, a possible solution is based on the substitution of the collective communcations by a pool of nonblocking, point-to-point communications. The advantages of such schemes are twofold: on one hand, the number of packets may be reduced. Moreover, communications and calculation may overlap, thus allowing to save the time used for communications, which is not the case for the traditional approach. This scheme has been already implemented in BigDFT for the exact exchange operator calculation, providing really encouranging performances. The generalisation to the other operations of the code is underway. To this aim, an important part of the present work is to stabilize the parallelisation scheme to be used for the O(N) version of BigDFT code, which is under intense development at present. Collective variable communications (MPI_ALLTOALLV) will be tested against non-blocking send-receive strategy. An important point will be the identification of the crossover point between the two strategies as a function of system-dependent parameter (wavefunction size and localisation).

**References**


1. W. Kohn and L. J. Sham, Phys. Rev. **140**, A1133 (1965)
2. L. Genovese et al., J. Chem. Phys. 125, 074105 (2006)
3. L. Genovese et al., J. Chem. Phys. 127, 054704 (2007)
4. L. Genovese et al, J. Chem. Phys. 129, 014109 (2008)
5. L. Genovese et al, J. Chem. Phys. 131, 034103 (2009)
6. L. Genovese et al, C. R. Mécanique 339, 2-3, 149-164, (2011)
7. S. Goedecker, A. Hoisie, *"Performance Optimization of Numerically Intensive Codes"*, SIAM publishing company, Philadelphia, USA 2001 (ISBN 0-89871-484-2)
8. S. Goedecker, M. Teter, J. Hutter, Phys. Rev. B **54**, 1703 (1996)
9. C. Hartwigsen, S. Goedecker and J. Hutter, Phys. Rev. B **58**, 3641 (1998)
10. M. Krack, Theor. Chem. Acc. **114**, 145 (2005)
11. S. Goedecker, *" Rotating a three-dimensional array in optimal positions for vector processing: Case study for a three-dimensional Fast Fourier Transform"*, Comp. Phys. Commun. **76**, 294 (1993)