6 SCIENTIFIC HIGHLIGHT OF THE MONTH

TDDFT in massively parallel computer architectures: the OCTOPUS project

Xavier Andrade[1*], Joseba Alberdi-Rodriguez[2,3], David A. Strubbe[4], Micael J. T. Oliveira[5], Fernando Nogueira[5], Alberto Castro[6], Javier Muguerza[3], Agustin Arruabarrena[3], Steven G. Louie[4], Alán Aspuru-Guzik[1], Angel Rubio[2,7] and Miguel A. L. Marques[8]

[1] Department of Chemistry and Chemical Biology, Harvard University, 12 Oxford Street, Cambridge, MA 02138, USA

[2] Nano-Bio Spectroscopy Group and ETSF Scientific Development Centre, Departamento de Física de Materiales, Centro de Física de Materiales CSIC-UPV/EHU and DIPC, University of the Basque Country UPV/EHU, Av. Tolosa 72, 20018 Donostia/San Sebastián, Spain

[3] Deparment of Computer Architecture and Technology, University of the Basque Country UPV/EHU, M. Lardizabal 1, 20018 Donostia/San Sebastián, Spain

[4] Department of Physics, University of California, 366 LeConte Hall MC 7300, Berkeley, California 94720, USA and Materials Sciences Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, California 94720, USA

[5] Center for Computational Physics, University of Coimbra, Rua Larga, 3 004-516 Coimbra, Portugal

[6] Institute for Biocomputation and Physics of Complex Systems (BIFI), Zaragoza Center for Advanced Modelling (ZCAM), University of Zaragoza, Spain

[7] Fritz-Haber-Institut der Max-Planck-Gesellschaft, Berlin, Germany

[8] Université de Lyon, F-69000 Lyon, France and LPMCN, CNRS, UMR 5586, Université Lyon 1, F-69622 Villeurbanne, France

Abstract

OCTOPUS is a general-purpose density-functional theory (DFT) code, with a particular emphasis on the time-dependent version of DFT (TDDFT). In this article we present the ongoing efforts for the parallelisation of OCTOPUS. We focus on the real-time variant of TDDFT, where the time-dependent Kohn-Sham equations are directly propagated in time. This approach has a great potential for execution in massively parallel systems such as modern supercomputers with thousands of processors and graphics processing units (GPUs).

For harvesting the potential of conventional supercomputers, the main strategy is a multi-level parallelisation scheme that combines the inherent scalability of real-time TDDFT with a real-space grid domain-partitioning approach. A scalable Poisson solver is critical for the efficiency of this scheme. For GPUs, we show how using blocks of Kohn-Sham states

*xavier@tddft.org

provides the required level of data-parallelism and that this strategy is also applicable for code-optimisation on standard processors. Our results show that real-time TDDFT, as implemented in OCTOPUS, can be the method of choice to study the excited states of large molecular systems in modern parallel architectures.

1 Introduction

One of the main factors that has influenced the wide adoption of first principles electronic-structure methods in Physics and Chemistry was the fast growth of the capabilities of computers dictated by Moore's law [1]. This was combined with the development of new algorithms and software capable of exploiting these capabilities. Thanks to these advances, molecular or solid-state systems containing several thousands of atoms are now accurately modelled using supercomputers.

During the last years, however, we have witnessed a remarkable change in computer architectures. Before, the ever-increasing transistor density was directly translated into an increase of the speed and capabilities of a single processor core. However, problems related to efficiency, power consumption and heat dissipation forced a change of paradigm. So the trend is now to use the extra transistors to provide more processing elements. This is reflected in today's supercomputers, where the number of processor cores is constantly increasing while the capabilities of each processing element are progressing much slower. The same is true for personal computing, the parallelism can be obtained via multi-core central processing units (CPU), but also generally-programmable graphics processing units (GPU). In fact, GPUs effectively convert a desktop computer into a massively parallel computer.

Unfortunately, the task of using efficiently all the available parallel units concurrently is left to the application programmer. This is a very complex task, and presents a real challenge to programmers in general and to scientists in particular. Of course, as new parallel programming paradigms are being introduced, scientific codes also have to evolve. Some of the methods that we can currently use can be adapted to parallel architectures, but many popular scientific computing techniques or algorithms might have to be replaced by others that are more efficient in these new environments. In exchange, this change in computational paradigm offers us the possibility of studying larger systems under more realistic conditions and using more accurate methods than it was possible before.

In this article, we show how time-dependent density functional theory (TDDFT) [2] in real time can be a very competitive approach to study the excited states of electronic systems in massively parallel architectures, especially when combined with a spatial grid representation. In fact, real-time TDDFT is a versatile method to model the response of an electronic system (molecular or crystalline [3]) to different kinds of perturbations. It is useful to calculate properties like optical absorption spectra [4, 5], non-linear optical response [6, 7], circular dichroism [8, 9], van der Waals coefficients [10], Raman intensities [11], etc. The numerical advantage of real-time TDDFT is that the propagator preserves the orthogonality of the states [12]. In practice, this allows us to propagate each one of the states in an independent manner, which is ideal for parallelisation. Since the method does not require expensive orthogonalisation steps, the

numerical cost scales with the square of the size of the system and not cubically as many other methods [13].

Here, we present a summary of the work that has been made during the last years on the optimisation and parallelisation of the code OCTOPUS [14, 15], in order to profit from state-of-the-art high-performance computing platforms, from GPUs to supercomputers. This code is a widely used tool to perform real-time TDDFT simulations. It uses a real-space grid that provides an accurate and controllable discretisation that also allows for an efficient parallelisation by domain decomposition.

Directly following (Section 2), we give a brief description of OCTOPUS and its features. Then (Section 3), we focus our discussion in the parallelisation of OCTOPUS: the various parallelisation modes and how they can be combined depending on the system characteristics and the available processors. We pay special attention to the crucial bottleneck that can be the solution of Poisson's equation. Then we describe how we tackled the GPU parallelisation (Section 4). Finally (Section 5), we present some scalability results to illustrate all the efforts described in the previous sections.

2 Octopus features

2.1 Theory

OCTOPUS was originally written to solve the equations of density functional theory (DFT) in its ground-state [16] and time-dependent [2] forms. In particular, and like the vast majority of DFT applications, we use the Kohn-Sham (KS) [17] formulation of DFT, which leads to a coupled set of single-particle equations whose solution yields the many-body electronic density $n(\boldsymbol{r},t)$. For example, for the time-dependent case these equations read (atomic units are used throughout this article)

$$i\frac{\partial}{\partial t}\varphi_i(\boldsymbol{r},t) = \left[-\frac{1}{2}\nabla^2 + v_{\text{ext}}(\boldsymbol{r},t) + v_{\text{Hartree}}[n](\boldsymbol{r},t) + v_{\text{xc}}[n](\boldsymbol{r},t)\right]\varphi_i(\boldsymbol{r},t) \tag{1}$$

$$n(\boldsymbol{r},t) = \sum_{i}^{\text{occ}} |\varphi_i(\boldsymbol{r},t)|^2 \tag{2}$$

where $\varphi_i(\boldsymbol{r},t)$ are the single-particle KS states (also called KS orbitals), $v_{\text{ext}}(\boldsymbol{r},t)$ is the time-dependent external potential that can be the potential generated by the nuclei, a laser field, etc.; $v_{\text{Hartree}}[n](\boldsymbol{r},t)$ is the Hartree potential that describes the classical mean-field interaction of the electron distribution; and $v_{\text{xc}}[n](\boldsymbol{r},t)$ is the exchange-correlation (xc) potential that includes all non-trivial many-body contributions.

It is true that the (time-dependent) KS equations are an exact reformulation of (time-dependent) quantum mechanics. However, the exact form of the xc functional is unknown and, therefore, has to be approximated in any practical application of the theory. In OCTOPUS different approximations for this term are implemented, from local and semi-local functionals to more sophisticated orbital dependent functionals, including hybrids [18] and the optimised effective potential approach [19]. Asymptotic correction methods are also implemented [20].

We note that the local and semi-local xc functionals in OCTOPUS were implemented as a separate component, Libxc [21]. This library is now completely independent of OCTOPUS and is used by several projects like APE [22], GPAW [23], and ABINIT [24]. Currently it contains around 180 functionals for the exchange, correlation, and kinetic energies belonging to the local-density approximation (LDA), the generalized-gradient approximation (GGA), the meta-GGA, and hybrid functional families. Functionals for systems of reduced dimensionality (1D and 2D) are also included.

OCTOPUS can also be used to study model systems of different dimensionalities (currently up to five-dimensional systems are supported). For this, an arbitrary external potential can be given by the user by directly including its formula in the input file. This model is extremely useful, e.g., to study reduced-dimensional systems of interacting electrons in time-dependent fields [25–28]. In fact, the Schrödinger equation describing two electrons interacting in 1D is equivalent to a Schrödinger equation of one independent electron in 2D. In the same way, the problem of two electrons interacting in 2D can be mapped to the problem of one electron in a 4D space.

Another recent incorporation in the code is the possibility of performing multi-scale modelling by combining electronic systems with complex electrostatic environments. For example, this has been used to simulate a molecule placed between two metallic plates at a certain voltage bias [29]. Multiscale QM/MM calculations can be performed as well [30].

Besides ground-state and real-time TDDFT, OCTOPUS can do other types of calculations. It can perform Ehrenfest-TDDFT non-adiabatic molecular dynamics [14, 31] and adiabatic molecular dynamics based on the modified Ehrenfest scheme [32, 33], which inherits the scalability properties of real-time TDDFT, or the standard Born-Oppenheimer and Car-Parrinello [34] schemes. Different response properties in TDDFT [35] can also be obtained using linear-response formalisms like Casida [36], or the density-functional perturbation theory/Sternheimer approach [37–43]. OCTOPUS can also do quantum optimal-control calculations [44–46] and real-time quantum transport calculations [47]. OCTOPUS can generate the DFT data required for GW and Bethe-Salpeter calculations using the BerkeleyGW code [48].

2.2 Grids

OCTOPUS uses a real-space grid discretisation to represent fields such as the Kohn-Sham states and the electronic density. Each function is represented by its value over an array of points distributed in real space. Differential operators are approximated by high-order finite-difference methods [49] while integration is performed by a simple sum over the grid point coefficients.

The real-space grid approach does not impose a particular form for the boundary conditions, so it is possible to model both finite and periodic systems directly. Moreover, in OCTOPUS the grid boundaries can have an arbitrary shape, avoiding unnecessary grid points. For example, for molecular calculations the default box shape corresponds to the union of spheres centred around the atoms (see Fig. 1 for an example).

One of the main advantages of the real-space grid approach is that it is possible to systematically control the quality of the discretisation. By reducing the spacing and increasing the size of the box, the error is systematically decreased, and can be made as small as desired, at the cost of an

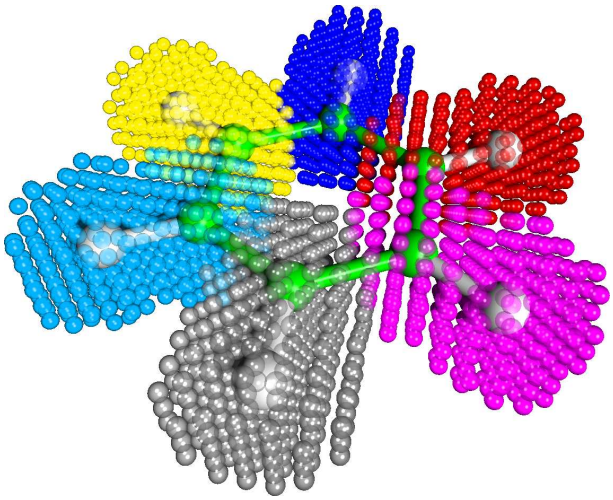

Figure 1: Example of the grid of OCTOPUS for a benzene molecule. The different colours represent different processors in a domain decomposition parallelisation.

increased computational cost. This is of particular significance for response properties [42, 50].

While the real space scheme results in a large number of discretisation coefficients when compared with localised basis set representations, the discretised Hamiltonian is very sparse. In fact, the number of non-zero components depends linearly on the number of coefficients. Moreover, the Hamiltonian only requires information from near neighbours, which is advantageous for parallelisation and optimisation.

Finally, since the description of the core regions is expensive with a uniform-resolution discretisation, in OCTOPUS the ion-electron interaction is usually modelled using norm-conserving pseudo-potentials [51]. At the moment, the code can read pseudo-potentials in several formats: the SIESTA format [52], the Hartwigsen-Goedecker-Hutter format [53], the Fritz-Haber format [54] and its Abinit version [24], and the Quantum Espresso universal pseudo-potential format [55]. Relativistic corrections, like spin-orbit coupling, can also be included by using relativistic pseudopotentials [56, 57].

2.3 The software package

The source code of OCTOPUS is publicly available under the GNU public license (GPL) v.2.0. This allows anyone to use the code, study it, modify it and distribute it, as long as these rights are retained for other users. We believe that this is something of particular importance for scientific work [58]. The code is written mainly in Fortran 95 with some parts in C and OpenCL [59]. Currently it consists of 180,000 lines of code (excluding external libraries).

Since the code is publicly available, it is essential to provide documentation so users can learn how to use it. The OCTOPUS website[1] contains a user manual and several tutorials that teach users how to perform different types of calculations, including some basic examples. Additionally, all input variables have their own documentation that can be accessed through the website or from a command line utility. A mailing list is also available, where users can get help with specific

[1] `http://tddft.org/programs/octopus/`

questions about OCTOPUS from the developers or other users.

One of the most important points in developing a scientific code is to ensure the correctness of the results. When a new feature is implemented, the developers validate the results by comparing them with known results from other methods and other codes. To ensure that future changes do not modify the results, we use an automated system (BuildBot [60]) that runs the code periodically and compares the results with reference data. A short set of tests is executed each time a change is made in OCTOPUS while a long one is executed every day. The tests are run on different platforms and with different compilation options, to ensure that the results are consistent for different platforms. Users should also run the testsuite to validate the build on their machine before running real calculations.

To avoid users inadvertently using parts of the code that are being developed or have not being properly validated, they are marked as "Experimental." These experimental features can only be used by explicitly setting a variable in the input file. In any case, users are expected to validate their results in known situations before making scientific predictions based on OCTOPUS results.

3 Parallelisation

In order to take advantages of modern day architectures, OCTOPUS uses a hybrid parallelisation scheme. This scheme is based on a distributed memory approach using the message passing interface (MPI) library for the communication between processes. This is combined with fine-grained parallelism inside each process using either OpenCL or OpenMP.

The MPI parallelisation is mainly based on a tree-based data parallelism approach, even if some steps have already been done in order to take advantage of task parallelism. The main piece of data to be divided among processes are the KS states, an object that depends on three main indices: a combined k-point and spin index, the state index, and the space coordinate. Each one of these indices is associated with a data-parallelisation level, where each process is assigned a section of the total range of the index. Note that since the k-point and spin index are both symmetry related quantum numbers, it is convenient to combine them in a unique index that labels the wave-functions.

This multi-level parallelisation scheme is essential to ensure the scaling of real-time TDDFT. As the size of the system is increased, two factors affect the computational time: first, the region of space that needs to be simulated increases, and second, the number of electrons increases. By dividing each of these degrees of freedom among processors, multi-level parallelisation ensures that the total parallel efficiency remains constant as we increase the system size and the number of processors.

3.1 Parallelisation in K-points and spin

For independent particles, the Schrödinger Hamiltonian can be exactly partitioned according to the k-point and spin labels. This means that each one of the subproblems, *i.e.* for each k-point and spin label, can be solved independently of the others, reducing thereby the dimension of

the Hamiltonian and the computational complexity. Mathematically, in (time-dependent) DFT this is no longer true as the subproblems are mixed by the density. However, a large part of the numerical solution can still be partitioned effectively, making the parallelisation in k-points and spin very efficient as little communication is required. Such a scheme does not always help for scaling, unfortunately. For example, for finite systems only a single k-point is used and in many cases we are interested in spin-unpolarised calculations. In this extreme case, there is absolutely no advantage in this parallelisation level.

3.2 Parallelisation in Kohn-Sham states

The following level of parallelisation regards the distribution of the KS states between processors. The problem is very different for ground-state DFT and for real-time TDDFT, so we discuss these cases separately.

For real-time TDDFT the propagation of each orbital is almost independent of the others, as the only interaction occurs through the time-dependent density. This again leads to a very efficient parallelisation scheme for time propagation that is only limited by the number of available KS states. In fact, when too few states are given to each process (assuming that no other parallelisation level is used), the cost of state-independent calculations starts to dominate (in particular the cost of solving the Poisson equation required to obtain the Hartree potential). As a rule of thumb, one should not have less than 4 states per process in order not to lose efficiency. Note that when ions are allowed to move during the propagation, a complementary parallelisation over atoms is used by OCTOPUS. This ensures that routines that calculate forces and that re-generate the atomic potential at each step do not spoil the very favourable scaling of TDDFT.

For ground-state calculations the parallelisation over states is more complex than for the time-propagation case, as the orthogonality constraint forces the diagonalisation procedure to explicitly mix different states. Regarding parallelisation, the most efficient eigensolver implemented in OCTOPUS turns out to be residual minimisation method – direct inversion in iterative subspace (RMM-DIIS) [61, 62] where the mixing of orbitals is restricted to two procedures: the Gram-Schmidt orthogonalisation [63] and the subspace diagonalisation. To parallelise these operations we use the parallel linear-algebra library Scalapack [64]. State-parallelisation of these procedures is particularly important as they involve matrices whose dimension is given by the number of KS states. Without state-parallelisation a complete copy of these matrices is kept by each process, which can easily lead to memory problems for systems with thousands of atoms.

3.3 Parallelisation in domains

The final level of MPI parallelisation consists in assigning a certain number of grid points to each process. In practice, the space is divided in domains that are assigned to each process. As the finite difference operators, like the Laplacian, only require the values of neighbouring points, the (small) boundary regions between domains need to be communicated between processors. In OCTOPUS this is done asynchronously, which means that the boundary values are copied between processors while the Laplacian calculation is done over the points in the central part

of the domain that do not require this information. This decreases substantially the problems related to latency, thereby improving the scaling with the number of processes.

Note that also the calculation of integrals requires a communication step to add the partial sum over each domain, an operation known as a reduction. The strategy to reduce the cost of this communication step is to group reductions together, as the cost of reductions of small vectors of data is dominated by the latency in the communication.

An important issue in domain parallelisation is selecting which points are assigned to each processor. This task, known as grid partitioning, is not trivial for grids with an arbitrary shape. Not only the number of points must be balanced between processors but also the number of points in the boundary regions must be minimised. OCTOPUS relies on external libraries for this task, with two currently supported: METIS [65] and ZOLTAN [66]. An example of the grid partitioning scheme is shown in Fig. 1.

Certainly the communication cost of the domain parallelisation is considerably higher than for the other schemes. It is also the most complicated to implement. However, once the basic grid operations are implemented it is almost transparent for developers to write parallel code.

3.4 Parallelisation of the Poisson solver

In order to obtain the Hartree potential from the electronic density, we solve the Poisson equation

$$\nabla^2 v_{\mathrm{Hartree}}(\boldsymbol{r}) = -4\pi n(\boldsymbol{r}) \tag{3}$$

When performing real-time propagation, the solution of this equation becomes the main bottleneck when thousands of processors are used [67]. The reason is clear: as there is only one Poisson equation to solve (regardless of the number of KS states), domain partitioning is the only available scheme to parallelise this operation. Unfortunately, the solution of the Poisson equation is highly non-local, mixing information from all points, making it unsuitable for the domain decomposition approach that relies on space locality. This can be easily seen from the integral form of Eq. (3)

$$v_{\mathrm{Hartree}}(\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}' \frac{n(\boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|} \tag{4}$$

Fortunately, several solutions to this problem exist, given the wide use of the Poisson equation in different fields. In OCTOPUS this parallelisation level is handled in a special way, and all processes are used independently of the distribution scheme. We found two particularly efficient approaches.

The first one is the use of fast Fourier transforms (FFT) to evaluate (4) in reciprocal space, a very fast method when running on a single processor. However, FFTs are not easy to parallelise. Additionally, the standard FFT approach yields a Poisson potential with periodic boundary conditions, so some modifications have to be made to obtain the free-space solution. Two different FFT-based solvers are available in OCTOPUS: (i) the software package provided by Genovese *et al.* [68] that uses an interpolating scaling functions (ISF) approach and the parallel FFT routine by Gödecker [69]; (ii) the parallel fast Fourier transform (PFFT) library [70] combined with the free-space Coulomb kernel proposed by Rozzi *et al.* [71].

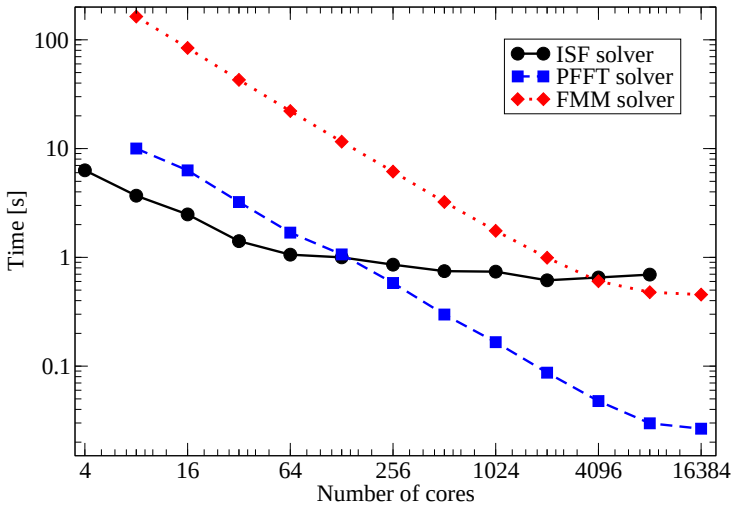

Figure 2: Time required to solve the Poisson equation for different solvers implemented in OCTOPUS. Grid of size 15^3 on a IBM Blue Gene/P system.

The second parallel Poisson solver we tested is the fast multipole method (FMM) [72]. This is an approximate method that reduces the complexity of evaluating Eq. (4) by using a multipole expansion to approximate the far-field effect from several charges into a single term. Our implementation is based on the FMM library [73]. This library is designed to calculate the interaction between point charges, therefore, we have to assume that the density in the grid corresponds to an array of point charges and then calculate a correction for the interaction between neighbouring points [74]. This correction term, essential to obtain the necessary accuracy, has the form of a finite-differences operator, and is therefore simple to evaluate in our framework.

In Fig. 2, we compare the time required to solve the Poisson equation in a parallelepiped box of size 15^3. As we can see, the ISF method is faster for small number of processors, while PFFT and FMM become more competitive as we increase their number. The crossover point is quite low with PFFT method (128 processors) and higher with FMM (4096 processors). More details about the parallel solution of the Poisson equation in OCTOPUS can be found in Ref. [74].

4 GPU parallelisation and code optimisation

Graphical processing units were initially designed for generating graphics in real time. They are massively parallel processors with hundreds or thousands of execution units. Given their particular architecture, GPUs require code written in a explicitly parallel language. The OCTOPUS support for GPUs is based on OpenCL, an open and platform-independent framework for high-performance computing on parallel processors. OpenCL implementations are available for GPUs from different vendors, for multi-core CPUs, and for dedicated accelerator boards. Since the OpenCL standard only defines a C interface, we have developed our own interface to call OpenCL from Fortran. This interface is currently available as an independent library, FortranCL [75].

For optimal performance, GPUs must process several streams of independent data simultaneously. This implies that performance critical routines must receive a considerable amount of data

on each call. To do this, the GPU optimisation strategy of OCTOPUS is based on the concept of a block of states, *i.e.*, a small group of KS states. Performance-critical routines are designed to operate over these blocks. This approach provides a larger potential for data parallelism than routines that operate over a single state at a time. It turns out that this strategy also works well for CPUs with vectorial floating units, where parallelisation is based on the OpenMP framework combined with explicit vectorisation using compiler directives.

For both GPUs and CPUs, memory access tends to be the main limitation to the performance of OCTOPUS. Working with blocks of orbitals improves memory access, provided that the coefficients are ordered in memory by the state index, so that load and stores are done from/to sequential addresses. However, increasing the KS block size can have a negative effect in memory access as larger data sets are less likely to benefit from cache memory. This is particularly critical for CPUs that depend much more than GPUs on caching for optimal performance.

One routine that can greatly benefit from caching is the application of the finite-difference Laplacian operator required for the kinetic-energy term. This is also an important operation as it represents a considerable part of the execution time of OCTOPUS. We devised an approach to improve cache utilisation by controlling how grid points are ordered in memory, *i.e.*, how the three-dimensional grid is enumerated. The standard approach is to use a row-major or column-major order which leads to neighbouring points being often allocated in distant memory locations. Our approach is to enumerate the grid points based on a sequence of small cubic grids, so close spatial regions are stored close in memory, improving memory locality for the Laplacian operator. The effect of this optimisation can be seen in Fig. 3. For the CPU with the standard ordering of points, the performance decreases as the block size is increased, while by optimising the grid order the parallelism exposed by a larger block size allows a performance gain of approximately 40%. For the GPU the effect of the optimisation is less dramatic but still significant.

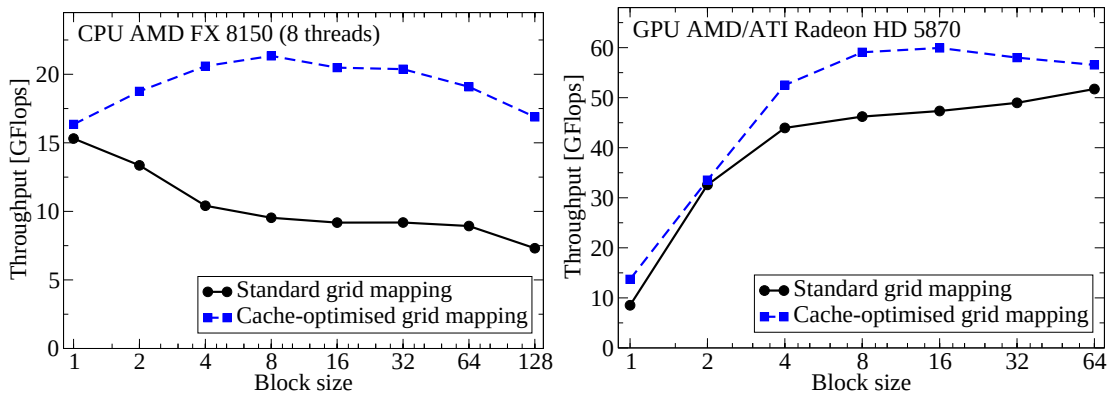

Figure 3: Effect of the optimisation of the grid mapping for data locality in the numerical throughput of the Laplacian operator as a function of the size of the KS states block. *Left* computations with an AMD FX 8150 CPU (8 threads). *Right*: computations with an AMD/ATI Radeon HD 5870 GPU.

In Fig. 4 we show a comparison of the numerical throughput of the GPU and CPU implementations of the Laplacian operator as a function of the size of the KS states block. It can be seen that the use of blocks of KS states represents a significant numerical performance gain with

respect to working with one state at a time. This is particularly important for GPUs where performance with a single state is similar to the CPU one but can be tripled by using a block of size 16 or 32. The same conclusion can be reached by looking at the numerical throughput of the orbital propagation and the total time required for a TDDFT iteration (see Fig. 5 and Table 1). The use of GPUs gives quite spectacular improvements, with a total iteration time being decreased by more than a factor of 6 with respect to the optimised multi-threaded CPU implementation. In fact, the propagation of the KS orbitals is 8 times faster in the GPU, but currently the total speed-up is limited by the Poisson solver that is executed on the CPU. These results mean that using a single GPU OCTOPUS could obtain the absorption spectrum of the C_{60} molecule in about one hour. More details about the GPU implementation in OCTOPUS can be found in Refs. [76, 77].

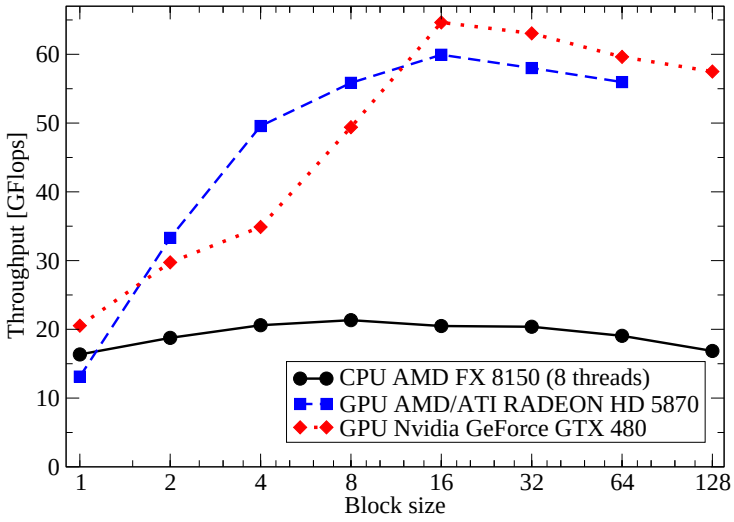

Figure 4: Numerical throughput of the OCTOPUS implementation of the Laplacian operator for different processors as a function of the number of KS states in a block. Spherical grid with 5×10^5 points. For each point the grid ordering has been adjusted for optimal cache usage.

Processor	Time per step [s]
CPU AMD FX 8150 (8 threads)	9.45
GPU AMD/ATI Radeon HD 5870	1.86
GPU NVIDIA GeForce GTX 480	1.72

Table 1: OCTOPUS propagation time per step for different processors. C_{60} molecule with a grid of spheres of radius 10 a.u. around each atom and spacing 0.375 a.u.

5 Scalability in massively parallel systems

In this section we show how all the improvements in algorithms, parallelisation, and optimisation are combined in the simulation of large electronic systems. As a benchmark we used portions of the spinach photosynthetic unit [78] with 180, 441, 650, 1365 and 2676 atoms containing several chlorophyll units. As these are molecular systems, only the state and domain decomposition parallelisation levels are used. We emphasise that these are real-world examples, that were not

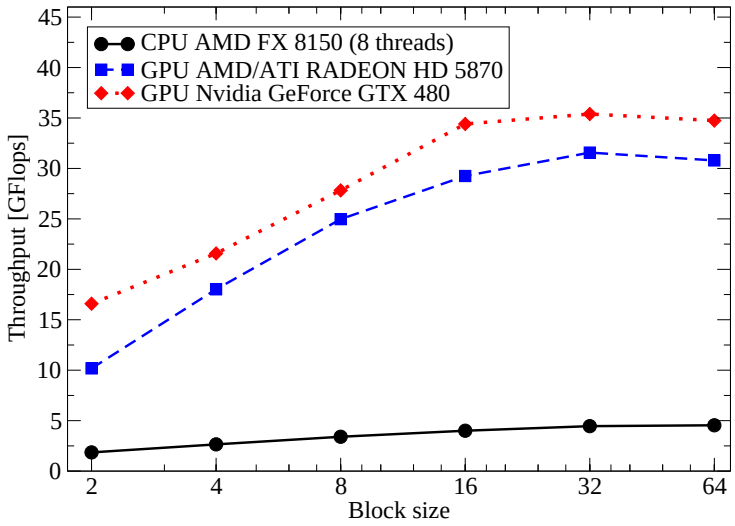

Figure 5: Numerical throughput of the KS state propagation in OCTOPUS for different processors as a function of the size of the KS states blocks. C_{60} molecule with a grid of spheres of radius 10 a.u. around each atom and spacing 0.375 a.u.

tweaked in order to artificially improve throughput or scalability.

We used different supercomputers for our benchmarks, as using more than one machine ensures a better portability of the code and inherent scalability of the algorithm. The first one is Curie, a supercomputer that belongs to the French Commissariat à l'Energie Atomique (CEA) and that consists of Intel Xeon processors interconnected by an Infiniband network. In total it has 11,520 cores available to users. The second platform is the IBM Blue Gene/P, a design based on low power processors (IBM PowerPC 450) interconnected by two networks, one with a toroidal topology and the other with a tree topology. The Blue Gene/P is a challenging architecture as it has a very limited amount of memory per core (512 MiB). Because of this, inside each node OpenMP parallelisation is used as this guarantees a minimal replication of data. Compiler directives are also used to profit from the vectorial floating-point units. The Blue Gene/P calculations presented in this work were performed in the 16,384-core supercomputer of the Rechenzentrum Garching of the Max Planck Society, Germany.

We benchmark separately the two main tasks performed by OCTOPUS. The first concerns the calculation of the ground-state of the system, while the second regards real-time TDDFT runs. We stress again that the parallelisation of the first task is considerably more difficult than of the second. However, the parallelisation of the time-dependent runs is much more important, as this consumes much more computational time than the ground-state runs.

Note that OCTOPUS has to perform several initialisations (create the grid, reserve memory, prepare the MPI environment, etc.) before starting the actual simulations. This time is usually negligible compared to the total simulation time. Our results are therefore measured in terms of elapsed wall-clock time per self-consistent iteration (for the ground-state) or per time step (for the time-dependent runs). The total execution time of a real-world simulation is basically proportional to these numbers.

5.1 Ground-state calculations

We start by showing scalability tests for ground-state calculations for a system of 180 and 441 atoms executed on a Blue Gene/P system. In order to measure the ground-state iteration time, 10 self-consistency cycles were run and the average time is shown in Fig. 6. We also show the parallel speed-up, the relative performance with respect to the run with the smallest number of processors. Excellent scalability is achieved up to 256-512 cores, and up to 4096 cores the time per iteration decreases as the number of processors is increased. As expected, scaling improves for systems with a larger number of atoms.

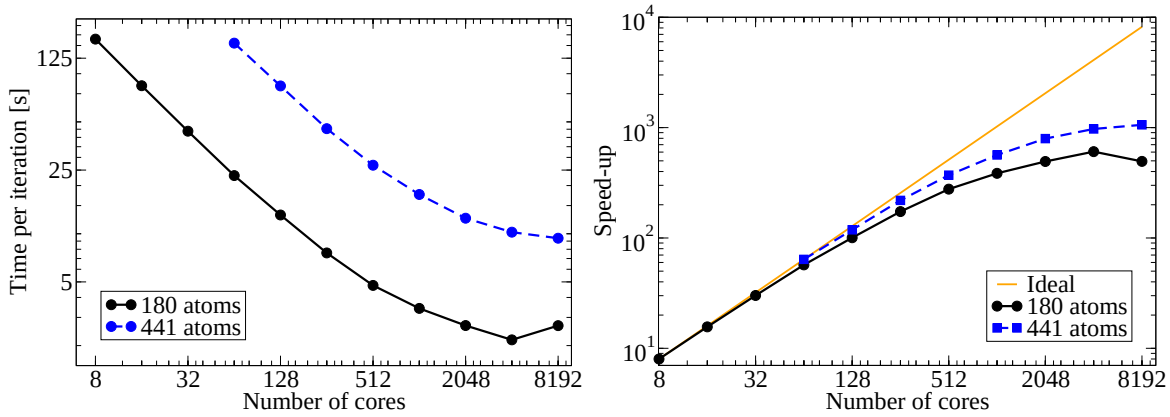

Figure 6: Ground-state calculation on a Blue Gene/P system for different number of atoms. *Left:* Wall-clock time per self-consistency iteration. *Right*: Parallel speed-up compared with the ideal case.

Recall that all runs introduced here were performed in symmetric multiprocessing (SMP) mode, combining inner-loop OpenMP parallelisation with domain MPI parallelisation. This allows us not only to take advantage of the architecture of the CPU, but also to make better use of the (limited) memory available in each of the Blue Gene nodes. For instance, the run with 1024 cores was done launching 256 MPI processes, each of them with 4 OpenMP threads.

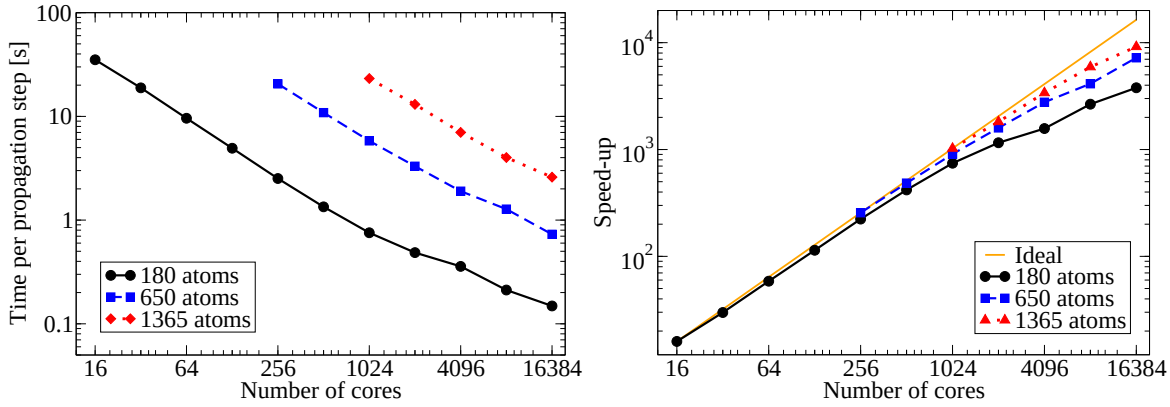

Figure 7: Real-time TDDFT propagation on a Blue Gene/P system for different numbers of atoms. *Left:* Wall-clock time per time step. *Right*: Parallel speed-up compared with the ideal case.

5.2 Real-time TDDFT calculations

In Fig. 7, we show the execution times and scaling for time-propagation runs for the systems of 180, 650 and 1365 atoms executed on a Blue Gene/P system also in SMP mode. We combine different ways to partition the cores between the domain and states parallelisation, selecting only the most efficient combination. We used a minimum of 16 cores with 180 atoms, 256 cores with 650 atoms and 1024 cores with 1365 atoms, as runs with fewer cores were not possible due to memory limitations. Almost perfect scaling was achieved up to 512 cores with the system of 180 atoms, and a remarkable speed-up of 3785 was obtained with 16384 cores, reaching an efficiency of 23% with 91 cores per atom. In the case of the system composed by 650 atoms the ideal speed-up is reached up to 2048 cores, while for 16384 cores a value of 7222 is achieved. Even better performance is obtained with the system of 1365 atoms, with a speed up of 9157 for 16384 cores. Thus, we can maintain an almost ideal speed-up with around 3 cores per atom, and acceptable speed-ups up to 50 cores per atom.

Benchmark tests also were done in the Curie supercomputer. The results are show in Fig. 8. The main advantage of this machine is that it has more powerful processors and more memory per core. However, the drawback is that the network is not as reliable as the Blue Gene one. The network is a shared resource and depends not only in the current run but also in all other running jobs. Consequently, only the minimum iteration time is shown here as an example of the best possible execution. Nevertheless, and confirming the scalability of the algorithm, remarkable speed-ups are achieved up to 8192 cores.

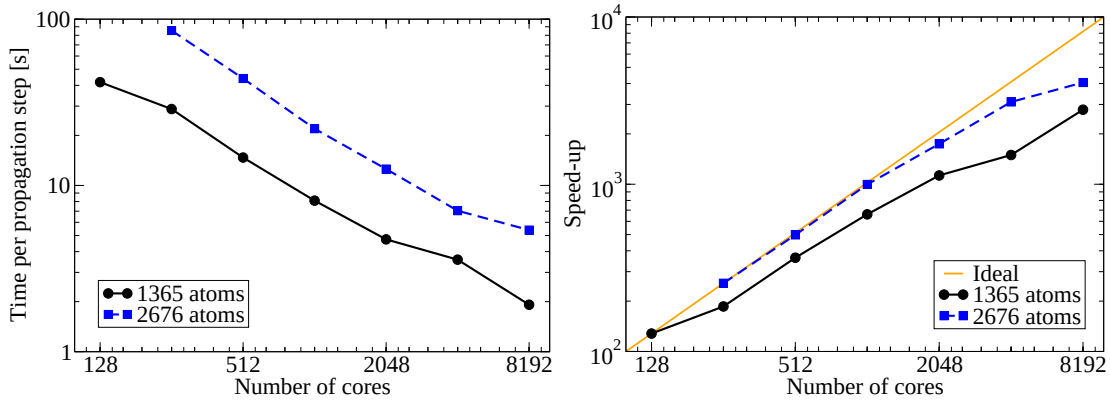

Figure 8: Real-time TDDFT propagation on the Curie system for different number of atoms. *Left:* Wall-clock time per time step. *Right*: Parallel speed-up compared with the ideal case.

We note that these tests were performed with the PFFT Poisson solver, which shows a great improvement when a large number of nodes are used when compared with the default ISF solver. The improvements made to the Poisson solver are reflected in the entire time-dependent iteration time. For large numbers of processes, the total iteration time is improved by up to 58% with respect to the ISF solver.

5.3 Combined MPI-GPU parallelisation

Octopus can also combine the MPI and OpenCL parallelisations in a hybrid approach to use

multiple GPUs. This has created some additional challenges when compared to the multiple-CPU parallelisation. First of all, as GPUs offer higher computational capabilities than CPUs, the time spent in communication becomes a larger fraction of the total execution time. In second place, communication times can become higher as transferring data between GPUs is a three-step procedure: first the data is copied to main memory using OpenCL, then it is sent to the other process using MPI; and finally it is copied to the memory of the second GPU using OpenCL. Still OCTOPUS can scale reasonably well when running on multiple GPUs, as it can be seen in Fig. 9. The speed-up for 8 GPUs is 5.7, an efficiency of 71%. Note that the main limitation to scalability in this case is the lack of a GPU accelerated parallel Poisson solver.

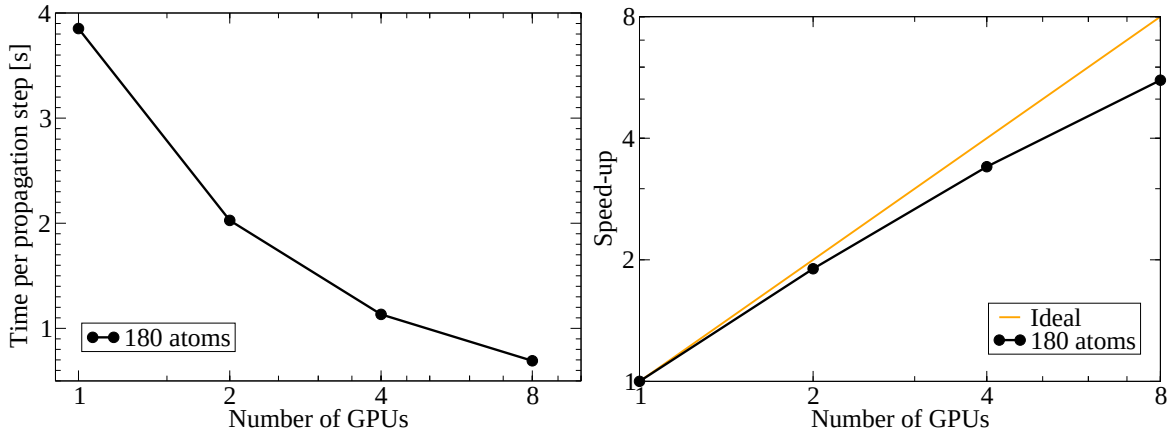

Figure 9: Scaling of time-propagation with multiple Nvidia Tesla M2090 GPUs. *Left:* Wall-clock time per time step. *Right*: Parallel speed-up compared with the ideal case.

6 Conclusions

In summary, our results show that real-time solution of TDDFT equations is a very parallelisable task. In our calculations we can scale with a reasonable efficiency to almost 100 cores per atom, which paves the way for TDDFT real-time simulations of systems with thousands of atoms. We would like to remark that all the calculations in this article were performed using simulation parameters that correspond to actual calculations, and that they were not tweaked to obtain better scalability. This parallelisability can also be exploited for efficient execution in parallel processors, including GPUs and multi-core CPUs with vectorial floating point units.

We also have shown a hybrid parallelisation scheme for execution on clusters of GPUs. This approach combines the high-level MPI parallelisation with low-level parallelisation based on OpenCL. For the moment our tests have been limited to a small number of GPUs, but clusters with thousands of GPUs are already available so certainly this approach will be developed with an eye on exaflop computing.

The current capabilities of OCTOPUS ensure that the excited-states properties of systems with thousands of atoms can be studied in the immediate future. The flexibility of real-time TDDFT method means that not only electronic linear response properties can be studied. In fact, non-linear phenomena or the effect of ionic motion in the response can all be tackled with this formalism. This is of particular interest, for example, in the study of quantum effects in

biological molecules, or in the interaction of matter with strong laser fields.

7 Acknowledgments

We would like to thank all those who have contributed to OCTOPUS development, in particular to F. Lorenzen, H. Appel, D. Nitsche, C. A. Rozzi, and M. Verstraete. We also acknowledge P. Garcia-Risueño, I. Kabadshow, and M. Pippig for their contribution in the implementation of the FMM and PFFT Poisson solvers in OCTOPUS.

We acknowledge the PRACE Research Infrastructure for providing access to Jugene at the Forschungszentrum Juelich and Curie at Commissariat à l'Energie Atomique. We also acknowledge the Rechenzentrum Garching of the Max Planck Society for access to their Blue Gene/P. Computational resources were also provided by the US Department of Energy at Lawrence Berkeley National Laboratory's NERSC facility.

XA and AA acknowledge hardware donations by Advanced Micro Devices, Inc. (AMD) and Nvidia Corp., and support from the the US NSF CDI program (PHY-0835713). MALM acknowledges support from the French ANR (ANR-08-CEXC8-008-01) and computational resources from GENCI (project x2011096017). JAR acknowledges the scholarship of the University of the Basque Country UPV/EHU. DAS acknowledges support from the US NSF IGERT and Graduate Research Fellowship Programs. SGL was supported by NSF Grant No. DMR10-1006184 and by the Director, Office of Science, Office of Basic Energy Sciences, Materials Sciences and Engineering Division, U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The authors were partially supported by the European Research Council Advanced Grant DYNamo (ERC-2010-AdG -Proposal No. 267374), Spanish Grants (FIS2011-65702-C02-01 and PIB2010US-00652), ACI-Promociona (ACI2009-1036), Grupos Consolidados UPV/EHU del Gobierno Vasco (IT-319-07), University of the Basque Country UPV/EHU general funding for research groups ALDAPA (GIU10/02), Consolider nanoTHERM (Grant No. CSD2010-00044) and European Commission projects CRONOS (280879-2 CRONOS CP-FP7) and THEMA (FP7-NMP-2008-SMALL-2, 228539)

References

[1] G. E. Moore, Electronics **38** (1965).

[2] E. Runge and E. K. U. Gross, Phys. Rev. Lett. **52**, 997 (1984).

[3] G. F. Bertsch, J.-I. Iwata, A. Rubio, and K. Yabana, Phys. Rev. B **62**, 7998 (2000).

[4] K. Yabana and G. F. Bertsch, Phys. Rev. B **54**, 4484 (1996).

[5] A. Castro, M. A. L. Marques, J. A. Alonso, and A. Rubio, Journal of Computational and Theoretical Nanoscience **1**, 231 (2004).

[6] A. Castro, M. A. L. Marques, J. A. Alonso, G. F. Bertsch, and A. Rubio, Eur. Phys. J. D **28**, 211 (2004).

[7] Y. Takimoto, F. D. Vila, and J. J. Rehr, J. Chem. Phys. **127**, 154114 (2007).

[8] K. Yabana and G. F. Bertsch, Phys. Rev. A **60**, 1271 (1999).

[9] D. Varsano, L. A. E. Leal, X. Andrade, M. A. L. Marques, R. di Felice, and A. Rubio, Phys. Chem. Chem. Phys. **11**, 4481 (2009).

[10] M. A. L. Marques, A. Castro, G. Malloci, G. Mulas, and S. Botti, J. Chem. Phys. **127**, 014107 (2007).

[11] R. Aggarwal, L. Farrar, S. Saikin, X. Andrade, A. Aspuru-Guzik, and D. Polla, Solid State Commun. **152**, 204 (2012).

[12] A. Castro, M. A. L. Marques, and A. Rubio, J. Chem. Phys. **121**, 3425 (2004).

[13] T. Helgaker, P. Jorgensen, and J. Olsen, *Molecular Electronic-structure Theory* (John Wiley & Sons Inc, 2012), ISBN 9780470017609.

[14] M. A. L. Marques, A. Castro, G. F. Bertsch, and A. Rubio, Comput. Phys. Commun. **151**, 60 (2003).

[15] A. Castro, H. Appel, M. Oliveira, C. A. Rozzi, X. Andrade, F. Lorenzen, M. A. L. Marques, E. K. U. Gross, and A. Rubio, Phys. Status Solidi (b) **243**, 2465 (2006).

[16] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).

[17] W. Kohn and L. J. Sham, Phys. Rev. **140**, A1133 (1965).

[18] A. D. Becke, J. Chem. Phys. **98**, 1372 (1993).

[19] S. Kümmel and J. P. Perdew, Phys. Rev. Lett. **90**, 043004 (2003).

[20] X. Andrade and A. Aspuru-Guzik, Phys. Rev. Lett. **107**, 183002 (2011).

[21] M. A. L. Marques, M. J. T. Oliveira, and T. Burnus (2012), submitted to Comput. Phys. Commun.

[22] M. J. T. Oliveira and F. Nogueira, Comput. Phys. Commun. **178**, 524 (2008).

[23] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, Phys. Rev. B **71**, 035109 (2005).

[24] X. Gonze, B. Amadon, P.-M. Anglade, J.-M. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Côté, et al., Comput. Phys. Commun. **180**, 2582 (2009).

[25] N. Helbig, I. V. Tokatly, and A. Rubio, Phys. Rev. A **81**, 022504 (2010).

[26] N. Helbig, J. I. Fuks, M. Casula, M. J. Verstraete, M. A. L. Marques, I. V. Tokatly, and A. Rubio, Phys. Rev. A **83**, 032503 (2011).

[27] J. I. Fuks, N. Helbig, I. V. Tokatly, and A. Rubio, Phys. Rev. B **84**, 075107 (2011).

[28] J. I. Fuks, A. Rubio, and N. T. Maitra, Phys. Rev. A **83**, 042501 (2011).

[29] R. Olivares-Amaya, M. Stopa, X. Andrade, M. A. Watson, and A. Aspuru-Guzik, J. Phys. Chem. Lett. **2**, 682 (2011).

[30] M. A. L. Marques, X. López, D. Varsano, A. Castro, and A. Rubio, Phys. Rev. Lett. **90**, 258101 (2003); X. López, M. A. L. Marques, A. Castro, and A. Rubio, J. Am. Chem. Soc. **127**, 12329 (2005).

[31] S. Meng and E. Kaxiras, J. Chem. Phys. **129**, 054110 (2008).

[32] J. L. Alonso, X. Andrade, P. Echenique, F. Falceto, D. Prada-Gracia, and A. Rubio, Phys. Rev. Lett. **101**, 096403 (2008).

[33] X. Andrade, A. Castro, D. Zueco, J. L. Alonso, P. Echenique, F. Falceto, and A. Rubio, J. Chem. Theo. Comput. **5**, 728 (2009).

[34] R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471 (1985).

[35] D. A. Strubbe, L. Lehtovaara, A. Rubio, M. A. L. Marques, and S. G. Louie, in *Fundamentals of Time-Dependent Density Functional Theory*, edited by M. A. L. Marques, N. T. Maitra, F. M. S. Nogueira, E. K. U. Gross, and A. Rubio (Springer Berlin / Heidelberg, 2012), vol. 837 of *Lecture Notes in Physics*, pp. 139–166.

[36] M. E. Casida, C. Jamorski, F. Bohr, J. Guan, and D. R. Salahub, *Optical Properties from Density-Functional Theory* (1996), chap. 9, pp. 145–163.

[37] S. Baroni, S. de Gironcoli, A. Dal Corso, and P. Giannozzi, Rev. Mod. Phys. **73**, 515 (2001).

[38] E. S. Kadantsev and M. J. Stott, Phys. Rev. B **71**, 045104 (2005).

[39] X. Andrade, S. Botti, M. A. L. Marques, and A. Rubio, J. Chem. Phys. **126**, 184106 (2007).

[40] S. Botti, A. Castro, X. Andrade, A. Rubio, and M. A. L. Marques, Phys. Rev. B **78**, 035333 (2008).

[41] S. Botti, A. Castro, N. N. Lathiotakis, X. Andrade, and M. A. L. Marques, Phys. Chem. Chem. Phys. **11**, 4523 (2009).

[42] F. D. Vila, D. A. Strubbe, Y. Takimoto, X. Andrade, A. Rubio, S. G. Louie, and J. J. Rehr, J. Chem. Phys. **133**, 034111 (2010).

[43] G. P. Zhang, D. A. Strubbe, S. G. Louie, and T. F. George, Phys. Rev. A **84**, 023837 (2011).

[44] C. Brif, R. Chakrabarti, and H. Rabitz, New J. Phys. **12**, 075008 (2010).

[45] K. Krieger, A. Castro, and E. K. U. Gross, Chem. Phys. **391**, 50 (2011).

[46] A. Castro and E. K. U. Gross, in *Fundamentals of Time-Dependent Density Functional Theory*, edited by M. A. L. Marques, N. T. Maitra, F. M. S. Nogueira, E. K. U. Gross, and A. Rubio (Springer Berlin / Heidelberg, 2012), vol. 837 of *Lecture Notes in Physics*, pp. 265–276, ISBN 978-3-642-23517-7.

[47] S. Kurth, G. Stefanucci, C.-O. Almbladh, A. Rubio, and E. K. U. Gross, Phys. Rev. B **72**, 035308 (2005).

[48] J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, Comp. Phys. Commun. **183**, 1269 (2012).

[49] J. R. Chelikowsky, N. Troullier, and Y. Saad, Phys. Rev. Lett. **72**, 1240 (1994).

[50] D. Rappoport, ChemPhysChem **12**, 3404 (2011).

[51] N. Troullier and J. L. Martins, Phys. Rev. B **43**, 1993 (1991).

[52] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, and D. Sánchez-Portal, J. Phys.: Condens. Matter **14**, 2745 (2002).

[53] C. Hartwigsen, S. Goedecker, and J. Hutter, Phys. Rev. B **58**, 3641 (1998).

[54] M. Fuchs and M. Scheffler, Comput. Phys. Commun. **119**, 67 (1999).

[55] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, et al., J. Phys.: Condens. Matter **21**, 395502 (19pp) (2009), URL `http://www.quantum-espresso.org`.

[56] A. Castro, M. A. L. Marques, A. H. Romero, M. J. T. Oliveira, and A. Rubio, J. Chem. Phys. **129**, 144110 (2008).

[57] M. J. T. Oliveira, F. Nogueira, M. A. L. Marques, and A. Rubio, J. Chem. Phys. **131**, 214302 (2009).

[58] D. C. Ince, L. Hatton, and J. Graham-Cumming, Nature **482**, 485 (2012).

[59] A. Munshi, ed., *The OpenCL Specification* (Khronos group, Philadelphia, 2009).

[60] `http://buildbot.net/`.

[61] P. Pulay, Chem. Phys. Lett. **73**, 393 (1980).

[62] G. Kresse and J. Furthmüller, Phys. Rev. B **54**, 11169 (1996).

[63] J. P. Gram, J. Reine Angew. Math **94**, 71 (1883); E. Schmidt, Math. Ann. **63**, 433 (1907).

[64] L. S. Blackford, J. Choi, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, et al., in *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)* (IEEE Computer Society, Washington, DC, USA, 1996), Supercomputing '96, ISBN 0-89791-854-1.

[65] G. Karypis and V. Kumar, Tech. Rep. (1995).

[66] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, Comput. Sci. Eng. **4**, 90 (2002).

[67] J. Alberdi-Rodriguez, *Analysis of performance and scaling of the scientific code Octopus* (LAP LAMBERT Academic Publishing, 2010), ISBN 978-3848418350, URL `http://nano-bio.ehu.es/files/alberdi_master.pdf`.

[68] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, and G. Beylkin, J. Chem. Phys. **125**, 074105 (2006).

[69] S. Goedecker, Comput. Phys. Commun. **76**, 294 (1993); S. Goedecker, M. Boulet, and T. Deutsch, Comput. Phys. Commun. **154**, 105 (2003).

[70] M. Pippig, in *Proceedings of the HPC Status Conference of the Gauss-Allianz e.V.* (2010).

[71] C. A. Rozzi, D. Varsano, A. Marini, E. K. U. Gross, and A. Rubio, Phys. Rev. B **73**, 205119 (2006).

[72] L. F. Greengard and V. Rokhlin, J. Comp. Phys. **73**, 325 (1987); L. F. Greengard and V. Rokhlin, *The Rapid Evaluation of Potential Fields in Three Dimensions* (Springer Press, Berlin, Heidelberg, 1988).

[73] I. Kabadshow and B. Lang, in *International Conference on Computational Science (1)* (2007), pp. 716–722.

[74] P. García-Risueño, J. Alberdi-Rodriguez, M. J. T. Oliveira, X. Andrade, I. Kabadshow, M. Pippig, A. Rubio, J. Muguerza, and A. Arruabarrena (2012), to be submitted; P. García-Risueño, Ph.D. thesis, Universidad de Zaragoza (2011), `http://nano-bio.ehu.es/files/garciarisueno_phd_thesis.pdf`.

[75] `http://code.google.com/p/fortrancl/`.

[76] X. Andrade and L. Genovese, in *Fundamentals of time-dependent density functional theory*, edited by M. A. L. Marques, N. T. Maitra, F. M. S. Nogueira, E. K. U. Gross, and A. Rubio (Springer Berlin / Heidelberg, 2012), Lecture Notes in Physics, pp. 401–413.

[77] X. Andrade and A. Aspuru-Guzik (2012), *To be submitted*.

[78] Z. Liu, H. Yan, K. Wang, T. Kuang, J. Zhang, L. Gui, X. An, and W. Chang, Nature **428**, 287 (2004).