A survey of the parallel performance and accuracy of Poisson solvers for electronic structure calculations

Pablo García-Risueño*[1,2,3], Joseba Alberdi-Rodriguez[4,5], Micael J. T. Oliveira[6], Xavier Andrade[7], Michael Pippig[8], Javier Muguerza[4], Agustin Arruabarrena[4], and Angel Rubio [5,9]

[1]Institut für Physik, Humboldt Universität zu Berlin, Zum grossen Windkanal 6, 12489 Berlin, Germany
[2]Institute for Biocomputation and Physics of Complex Systems BIFI, Universidad de Zaragoza C/ Mariano Esquillor, 50018 Zaragoza (Spain)
[3]Instituto de Química Física Rocasolano (CSIC), C/ Serrano 119, 28006 Madrid, Spain
[4]Dept. of Computer Architecture and Technology, University of the Basque Country UPV/EHU, M. Lardizabal, 1, 20018 Donostia/San Sebastián, Spain
[5]Nano-Bio Spectroscopy Group and European Theoretical Spectroscopy Facility, Spanish node, University of the Basque Country UPV/EHU, Edif. Joxe Mari Korta, Av. Tolosa 72, 20018 Donostia/San Sebastián, Spain
[6]Center for Computational Physics, University of Coimbra, Rua Larga, 3004-516 Coimbra, Portugal
[7]Dept. of Chemistry and Chemical Biology, Harvard University, 12 Oxford street, Cambridge, MA 02138, USA
[8]Dept. of Mathematics, Chemnitz University of Technology, 09107 Chemnitz, Germany
[9] Centro de Física de Materiales, CSIC-UPV/EHU-MPC and DIPC, 20018 Donostia/San Sebastián, Spain

October 20, 2013

Abstract

*risueno@physik.hu-berlin.de

We present an analysis of different methods to calculate the classical electrostatic Hartree potential created by charge distributions. Our goal is to provide the reader with an estimation on the performance —in terms of both numerical complexity and accuracy— of popular Poisson solvers, and to give an intuitive idea on the way these solvers operate. Highly parallelisable routines have been implemented in a first-principle simulation code (Octopus) to be used in our tests, so that reliable conclusions about the capability of methods to tackle large systems in cluster computing can be obtained from our work.

Keywords:
Hartree potential, charge density, Poisson solver, parallelisation, linear scaling, fast multipole method, parallel fast Fourier transform, interpolating scaling functions, conjugate gradients, multigrid.

■

Contents

1 Introduction

The electrostatic interaction between charges is one of the most important phenomena of physics and chemistry, which makes the calculation of the energy and potential associated to a distribution of charges one of the most common problems in scientific computing. The calculation of potentials created by pairwise interactions is ubiquitous in atomic and molecular simulations, and also in fields like quantum chemistry, solid state physics, fluid dynamics, plasma physics and astronomy, among others.

In particular, the electrostatic interaction is a key part in the density functional theory (DFT) [1, 2] and time-dependent density functional theory (TDDFT) [3] formulations of quantum mechanics. In DFT the many-body Schrödinger equation is replaced by a set of single particle equations, the Kohn–Sham equations, which include an effective potential that contains the electronic interaction. Such effective potential is usually divided into three terms: the Hartree potential, the exchange-correlation (XC) potential, and the external potential. The Hartree term corresponds to the classical electrostatic potential generated by the electronic charge distribution.

The calculation of the potential associated to a charge distribution is then an important step in many numerical implementations of DFT/TDDFT. In addition, the calculation of the electrostatic potential associated to a charge density can appear in other contexts in electronic structure theory, like the approximation of the exchange term [4, 5, 6], or the calculation of integrals that appear in Hartree–Fock [7, 8] or Casida [9] theories. It is understandable then, that the calculation of the electrostatic potential has received much interest in the recent past within the community of electronic structure researchers [10, 11, 12, 13, 14, 15, 16].

Since at present the complexity of the problems requires massively parallel computational platforms [17], every algorithm for a time-consuming task must not only be efficient in serial, but also needs to keep its efficiency when run in a very large number of processors (e.g. more than 10,000 CPUs). The electrostatic interaction is non-local, and thus the information corresponding to different points interacting with each other can be stored in different computing units, with a non-negligible time for data-communication among them. This makes the choice of the algorithm for the calculation of the Hartree potential critical, as different algorithms also have different efficiencies. Thanks to the efficiency offered by the current generation of solvers, the calculation of the Hartree potential usually contributes a minor fraction of the computational time of a typical DFT calculation. However, there are cases where the Poisson solver hinders the numerical performance of electronic structure calculations. For example, in parallel implementations of DFT it is common to distribute the Kohn–Sham orbitals between processors [18]; for real-time TDDFT and molecular dynamics in particular, this is a very efficient strategy [19, 20, 21]. Nonetheless, since a single Poisson equation needs to be solved independently on the number of orbitals, the calculation of the Hartree potential becomes an important bottleneck for an efficient parallelisation [22, 21] as predicted by Amdahl's law [23], if it is not optimally parallelised. Such a bottleneck also appears in other contexts of computational chemistry and physics, like Molecular Dynamics [24].

The objective of this article is, therefore, to analyse the relative efficiencies and accuracies of some of the most popular methods to calculate the Hartree potential created by charge distributions. Our purpose is to provide the reader with estimates on the features of this solvers that make it possible to choose which of them is the most appropriate for his/her electronic structure calculations.

We start by giving a brief theoretical introduction to the Poisson equation problem and on the different methods to solve it. Next, we discuss the details of our implementation and the parallel computers we use. Following, we present the results of our numerical experiments. We finish by stating our conclusions. More specific derivations and analysis are provided in the supporting information.

2 Theoretical background

In the context of quantum mechanics, the electrons and their electric charge are delocalised over space forming a continuous charge distribution $\rho(\boldsymbol{r})$. Such a charge density creates an electrostatic (Hartree) potential $v(\boldsymbol{r})$, which is given by [25]

$$v(\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}\,' \frac{1}{4\pi\epsilon_0} \frac{\rho(\boldsymbol{r}\,')}{|\boldsymbol{r} - \boldsymbol{r}\,'|} \ , \tag{1}$$

where ϵ_0 is the electrical permittivity of the vacuum, $1/4\pi$ in atomic units. When considering the electric interaction in a medium, it is possible to approximate the polarization effects by replacing ϵ_0 by an effective permittivity, ϵ. In the context of electronic structure calculations this effective permittivity is used, for example, in multiscale simulations where part of the system is approximated by a continuous polarisable medium [26].

In 3D, it is simple to show that equation 1 is equivalent to the Poisson equation [27, 28]

$$\nabla^2 v(\boldsymbol{r}) + \frac{\rho(\boldsymbol{r})}{\epsilon_0} = 0 \ . \tag{2}$$

In fact, this equation provides a convenient general expression that is valid for different dimensions and boundary conditions. For example, to study crystalline systems, when periodic boundary conditions are usually imposed. It is also possible to simulate a molecular system interacting with ideal metallic surfaces by choosing the appropriate boundary conditions [29, 30]. Both formulations of the problem, i.e. equations 1 and 2, are quite useful: while some methods to calculate the Hartree potential are based on the former, others rely on the latter.

In order to numerically calculate the electrostatic potential we need to discretise the problem. To this end, we use a grid representation, which changes the charge density and the electrostatic potential to discrete functions, with values defined over a finite number of points distributed over a grid. Such an approach is used in many electronic structure codes, even when another type of representation is used for the orbitals. The direct calculation of the potential would take $\mathcal{O}(N^2)$ operations, N being the total number of grid points. This is prohibitive for systems beyond a given size. Fortunately, there exist a variety of methods — either using equation 1 or 2 — that by exploiting the properties of the problem, reduce the cost to a linear or quasi-linear dependency with a negligible accuracy drop. For our survey, we have selected several parallel implementations of some of the most popular of these methods (fast Fourier transform, interpolating scaling functions, fast multipole method, conjugate gradients, and multigrid). In the next subsection we introduce these methods and give a brief account of their theoretical foundations and properties.

2.1 Fast Fourier transform

The Fourier transform (FT) is a powerful mathematical tool both for analytical and numerical calculations, and certainly it can be used to calculate the electrostatic potential created by a charge distribution represented in an equispaced grid by operating as follows. Let $f(\boldsymbol{r})$ be a function whose Fourier transform $\hat{f}(\boldsymbol{k})$ exists. By construction $f = f(\boldsymbol{r})$, and $\hat{f}^{-1}\big(\hat{f}(\boldsymbol{k})\big)(\boldsymbol{r}) = f(\boldsymbol{r})$. This expression, plus the convolution property of the Fourier transform, applied to equation 1, imply that

$$v(\boldsymbol{r}) = \hat{v}^{-1}\big(\hat{v}(\boldsymbol{k})\big)(\boldsymbol{r}) = \frac{1}{4\pi\epsilon_0}\hat{v}^{-1}\big(\hat{\rho}(\boldsymbol{k})/|\boldsymbol{k}|^2\big) \ , \tag{3}$$

where we have used that the Fourier transform of the function $1/|\boldsymbol{r}|$ is [28] $(\widehat{1/|\boldsymbol{r}|})(\boldsymbol{k}) = 1/|\boldsymbol{k}|^2 = 1/(k_x^2 + k_y^2 + k_z^2)$.

Since $\rho(\boldsymbol{r})$ is represented in discrete equispaced points $(r_{j,k,l})$ at the centre of cells whose volume equals Ω, the Fourier transform of $\rho(\boldsymbol{r})$, i.e. $\hat{\rho}(\boldsymbol{k})$, can be calculated using its discrete Fourier transform

$$\begin{aligned} \hat{\rho}(\boldsymbol{k}) \ &:= \ (2\pi)^{-3/2}\int \mathrm{d}\boldsymbol{r}\ \exp(-i\boldsymbol{k}\cdot\boldsymbol{r})\rho(\boldsymbol{r}) && (4) \\ &\simeq \ (2\pi)^{-3/2}\Omega\sum_{j,k,l}\rho(r_{j,k,l})\exp(-i(k_x j + k_y k + k_z l)) \ . && (5) \end{aligned}$$

The use of equation 4 in equation 3 results in a discretised problem, which requires the use of a discrete Fourier transform plus an inverse discrete Fourier transform. The expression of the potential in terms of discrete Fourier transforms allows the application of the efficient FFT technique [31] (see below for details), so the problem can be solved in $\mathcal{O}(N\log_2 N)$ steps.

It is to be stressed that the use of the FT automatically imposes periodic boundary conditions on the density, and therefore to the potential. When finite systems are studied, some scheme is required to avoid the interaction between periodic images. A simple way to solve this is to increase the size of the real-space simulation cell and set the charge density to zero in the new points. This moves the periodic replicas of the density away, thus decreasing their effect on the potential [28, 32]. Another strategy is to replace the $1/(\epsilon_0\boldsymbol{k}^2)$ factor of equation 3, known as the kernel of the Poisson equation, by a quantity that gives the free space potential in the simulation region. This modified kernel has been presented in reference [12] for molecules, one-dimensional systems, and slabs. This Coulomb cut-off technique is very efficient and easy to implement. In our fast Fourier transform method we combine the two approaches, doubling the size of the cell and using a modified kernel [28, 12]. This results in a potential that accurately reproduces the free space results. On the other hand, this method also imposes some constrains. For example, in the case of molecules, the cut-off is spherical and, therefore, it requires the enlarged cell to be always cubic, regardless of the shape of the original cell.

The power of the FFT method has made it part of many algorithms for the calculations of pairwise interactions. In the method summarized in this section, we treat the whole contribution to the potential with FFT. There exist, however, other schemes where only one part of the potential is calculated using FFT, e.g. the Particle Mesh Ewald methods

[33]. In these, the charge is split into an analytical component and a numerical component, being the contribution to the potential of the latter calculated through FFT. One celebrated method to do this for finite systems is presented in [32]. Other celebrated Poisson solvers are also based on Ewald methods. Among them, we can highlight the Particle-Particle Particle-Mesh method [34], which operates similarly, but evaluating the interaction between nearby neighbouring charges with direct summations.

In one dimension, the discrete Fourier transform of a set of n complex numbers f_k (f_k can be, for example, the values of ρ in a set of discrete points) is given by

$$F_l = \sum_{k=0}^{n-1} f_k \exp\left(-2\pi i \tfrac{kl}{n}\right) \qquad \text{for } l = 0, \ldots, n-1 \ , \tag{6}$$

with i being the imaginary unit. The inverse discrete Fourier transform is given by

$$f_k = \frac{1}{n} \sum_{l=0}^{n-1} F_l \exp\left(+2\pi i \tfrac{kl}{n}\right) \qquad \text{for } k = 0, \ldots, n-1 \ . \tag{7}$$

The definitions above enable computational savings using the fact that both the input and output data sets ($\rho(\boldsymbol{r})$ and $v(\boldsymbol{r})$) are real. Thus $F_{n-l} = F_l^*$, and we just need to calculate one half of the n discrete Fourier transforms.

Solving the Poisson problem using equations 6 and 7 would require $\mathcal{O}(N^2)$ arithmetic operations (with e.g. $N = n^3$), which would not represent any improvement over the cost of evaluating the potential directly. However, in 1965, J. W. Cooley and J. W. Tukey published an algorithm called fast Fourier transform (FFT) [31] that exploits the special structure of equation 6 in order to reduce the arithmetic complexity. The basic idea of the radix-2 Cooley-Tukey FFT is to split a discrete FT of even size $n = 2m$ into two discrete FTs of half the length; e.g., for $l = 0, \ldots, m-1$ we have

$$\begin{aligned} F_{2l} &= \sum_{k=0}^{m-1} f_k \exp\left(-2\pi i \tfrac{k2l}{n}\right) + f_{k+m} \exp\left(-2\pi i \tfrac{(k+m)2l}{n}\right) \\ &= \sum_{k=0}^{m-1} (f_k + f_{k+m}) \exp\left(-2\pi i \tfrac{kl}{m}\right) \ ; \end{aligned} \tag{8}$$

$$\begin{aligned} F_{2l+1} &= \sum_{k=0}^{m-1} f_k \exp\left(-2\pi i \tfrac{k(2l+1)}{n}\right) + f_{k+m} \exp\left(-2\pi i \tfrac{(k+m)(2l+1)}{n}\right) \\ &= \sum_{k=0}^{m-1} \exp\left(-2\pi i \tfrac{k}{n}\right) (f_k - f_{k+m}) \exp\left(-2\pi i \tfrac{kl}{m}\right) \ . \end{aligned} \tag{9}$$

If we assume n to be a power of two, we can apply this splitting recursively $\log_2 n$ times, which leads to $\mathcal{O}(n \log_2 n)$ arithmetic operations for the calculation of equation 6. There exist analogous splitting for every divisible sizes [35], even for prime sizes [36].

2.2 Interpolating scaling functions

This method was developed by Genovese *et al.* [14] for the BIGDFT code [37]. Formally it is based on representing the density in a basis of interpolating scaling functions (ISF) that arise in the context of wavelet theory [38]. A representation of ρ from $\rho_{j,k,l}$ can be efficiently built by using wavelets, and efficient iterative solvers for the Hartree potential v can be tailored, e.g. from ordinary steepest descent or conjugate gradient techniques [39]. A non-iterative and accurate way to calculate v [14] is to use the fast Fourier transform (FFT) in addition to wavelets. If we represent

$$\rho(\boldsymbol{r}) = \sum_{j,k,l} \rho_{j,k,l}\phi(x-j)\phi(y-k)\phi(z-l) \ , \tag{10}$$

where $\boldsymbol{r} = x, y, z$ and ϕ are interpolating scaling functions in one dimension, then equation 1 becomes

$$v_{m,n,o} := v(\boldsymbol{r}_{m,n,o}) = \sum_{j,k,l} \rho_{j,k,l} K(j,m;k,n;l,o) \ , \tag{11}$$

where

$$K(j,m;k,n;l,o) := \int_V \mathrm{d}\boldsymbol{r}\,'\frac{\phi_j(x')\phi_k(y')\phi_l(z')}{|\boldsymbol{r}_{m,n,o}-\boldsymbol{r}\,'|} \ , \tag{12}$$

and V indicates the total volume of the system. Due to the definition of the ISF, the discrete kernel K satisfies $K(j,m;k,n;l,o) = K(j-m;k-n;l-o)$, and therefore equation 11 is a convolution, which can be efficiently treated using FFTs (see the previous Section). The evaluation of equation 12 can be approximated by expressing the inverse of r in a Gaussian basis ($1/r \simeq \sum_k \omega_k \exp(-p_k r^2)$), which also enables efficient treatment. All this makes the order of this method $N \log_2 N$. Another important characteristic of the method is that it uses a kernel in equation 3 that yields an accurate free space potential without having to enlarge the cell.

2.3 Fast multipole method

The fast multipole method (FMM) was first proposed in 1987 for 2D systems [40], and it was soon extended to 3D problems [41]. Although only $\mathcal{O}(N)$ operations are necessary to calculate the electrostatic potential, the first FMM versions had big prefactors that in practice made the method competitive only for huge systems or low accuracy calculations [42]. After thorough research, it was possible to develop signficantly more accurate and efficient versions of the FMM [43, 44], making it a largely celebrated method [45].

The original FMM was devised to calculate the potential generated by a set of discrete point-like particles

$$v(\boldsymbol{r}_i) = \frac{1}{4\pi\epsilon_0} \sum_{j=1, j\neq i}^{N} \frac{q_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \ , \tag{13}$$

which is different from the charge-distribution problem that we are studying in this work. While extensions of the FMM to the continuous problem exist [46, 47, 48], in order to profit from the efficient parallel FMM implementations we have devised a simple scheme to recast the continuous problem into a discrete charge one without losing precision.

We assume that each grid point $r_{j,k,l}$ corresponds to a charge of magnitude $\Omega\rho_{j,k,l}$, where $\Omega = h^3$ (h being the grid spacing) is the volume of the space associated to each grid point (cell volume). Using the FMM we calculate at each point the potential generated by this set of charges, $v^{\text{FMM}}_{j,k,l}$. However, this is not equivalent to the potential generated by the charge distribution, and some correction terms need to be included (see Section S7 of the supporting information for the derivation of these corrections). The first correcting term (self-interaction term) comes from the potential generated at each point by the charge contained in the same cell

$$v^{\text{SI}}_{j,k,l} = 2\pi \left(\frac{3}{4\pi}\right)^{2/3} h^2 \rho_{j,k,l} \ . \tag{14}$$

Additionally, we apply a correction to improve the accuracy of the interaction between neighbouring points, which has the largest error in the point-charge approximation. This correction term is derived using a formal cubic interpolation of the density to a finer grid, obtaining a simple finite-differences-like term

$$\begin{aligned} v^{\text{corr.}}_{j,k,l} &= h^2\big(27/32 + (\alpha)2\pi(3/4\pi)^{2/3}\big)\rho_{j,k,l} \\ &+ (h^2/16)\big(\rho_{j-1,k,l} + \rho_{j+1,k,l} + \rho_{j,k-1,l} + \rho_{j,k+1,l} + \rho_{j,k,l-1} + \rho_{j,k,l+1}\big) \\ &- (h^2/64)\big(\rho_{j-2,k,l} + \rho_{j+2,k,l} + \rho_{j,k-2,l} + \rho_{j,k+2,l} + \rho_{j,k,l-2} + \rho_{j,k,l+2}\big) \ . \end{aligned} \tag{15}$$

Here α is a parameter to compensate the charge within the cell (j, k, l) that is counted twice. The final expression for the potential is

$$v_{j,k,l} = v^{\text{FMM}}_{j,k,l} + v^{\text{SI}}_{j,k,l} + v^{\text{corr.}}_{j,k,l} \ . \tag{16}$$

Now we give a brief introduction of the FMM algorithm. More detailed explanations on the FMM can be found in Refs. [40, 41, 49, 43]. To introduce the method we use spherical coordinates in what remains of this Section.

Consider a system of l charges $\{q_i,\ i = 1, \ldots, l\}$ located at points $\{(\tau_i, \alpha_i, \beta_i),\ i = 1, \ldots, l\}$ which lie in a sphere D of radius a and centre at $Q = (\tau, \alpha, \beta)$. It can be proved [43] that the electric field created by them at a point $P = (r, \theta, \phi)$ outside D is

$$v(P) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O^m_n}{(r')^{n+1}} Y^m_n(\theta', \phi') \ , \tag{17}$$

where $P - Q = (r', \theta', \phi')$ and

$$O^m_n = \sum_{i=1}^{l} q_i \tau_i^n Y^{-m}_n(\alpha_i, \beta_i) \ , \tag{18}$$

with $Y^m_n(\alpha, \beta)$ known functions (the spherical harmonics). If P lies outside of a sphere D_1 of radius $a + \tau$ (see Figure 1A) the potential of equation 17 can be re-expressed as

$$v(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \frac{M^k_j}{r^{j+1}} Y^k_j(\theta, \phi) \ , \tag{19}$$

where

$$M_j^k = \sum_{n=0}^{j} \sum_{m=-n}^{n} \tag{20}$$
$$\frac{O_{j-n}^{k-m} \; i^{|k|-|m|-|k-m|} \sqrt{(j-n-k+m)!(j-n+k-m)!}\sqrt{(n-m)!(n+m)!} \; \tau^n \; Y_n^{-m}(\alpha,\beta)}{\sqrt{(j-k)!(j+k)!}} \; .$$

Note that the "entangled" expression of the potential of equation 13, in which the coordinates of the point where the potential is measured and the coordinates of the charge that creates the potential are together, has been converted to a "factorised" expression, in which the coordinates of the point where we measure the potential are in terms $(Y_j^k(\theta,\phi)/r^{j+1})$ that multiply terms (M_j^k) which depend on the coordinates of the charges that create the potential. It is this factorisation which enables efficient calculation of the potential that a set of charges creates at a given point by using the (previously calculated) expression of the potential created by this set of charges at other points.

If the set of l charges described above is located inside a sphere D_Q of radius a with centre at $Q = (\tau, \alpha, \beta)$, where $\tau > 2a$ (see Figure 1B), then equation 17 implies that the potential created by these charges inside a sphere D_0 of radius a centred at the origin is given by

$$v(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} L_j^k \; r^j \; Y_j^k(\theta,\phi) \; , \tag{21}$$

where

$$L_j^k = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m \; i^{|k-m|-|k|-|m|} \sqrt{(n-m)!(n+m)!}\sqrt{(j-k)!(j+k)!} \; Y_{j+n}^{m-k}(\alpha,\beta)}{(-1)^n \; \tau^{j+n+1} \; \sqrt{(j+n-m+k)!(j+n+m-k)!}} \; . \tag{22}$$

The evaluation of the equations above requires truncation of the infinite sums to a given order, which can be chosen to keep the error below a given threshold. The equations 19 and 21 enable the efficient calculation of the potential experienced by every charge of the system due to the influence of the other charges. In order to calculate it, the system is divided into a hierarchy of boxes. Level 0 is a single box containing the whole system; level 1 is a set of 8 boxes containing level 0; and so on (a box of level $\mathcal{L}$ consists of 8 boxes of level $\mathcal{L}+1$). Different boxes at a given level do not contain common charges. The highest level $(\mathcal{N}_l)$ contains several charges (in our case, each lying in a grid point) in every box. The procedure to calculate the potential in all grid points can be summarised as follows:

- For every box in the highest box level $\mathcal{N}_l$ (smallest boxes), we calculate the potential created by the charges in that box using equation 17.
- We gather 8 boxes of level $\mathcal{N}_l$ to form every box of level $\mathcal{N}_l$-1. We calculate the potential created by the charges of the $(\mathcal{N}_l - 1)$-box using the potentials created by the eight $(\mathcal{N}_l)$-boxes that form it. To this end, we use equation 19.
- We repeat this procedure (we use equation 19 to get the potentials created by box $\mathcal{L}$-1 by using those of box $\mathcal{L}$) until all the levels are swept.

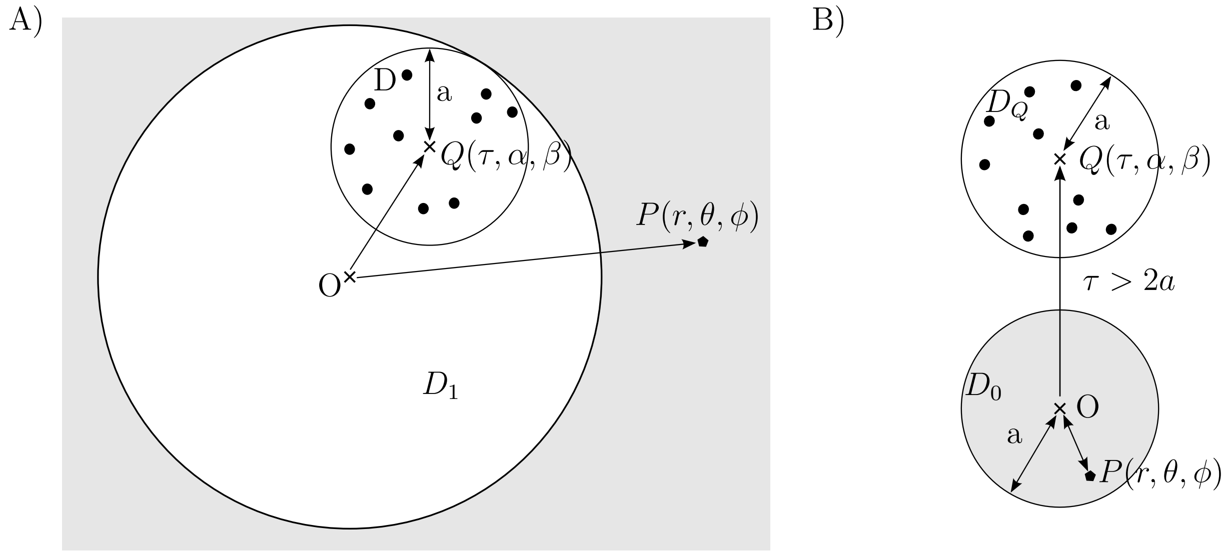


Figure 1: A) A set of point charges (black circles) inside a sphere D of radius a centred at $Q = (\tau, \alpha, \beta)$ creates a potential outside the sphere D_1 of radius $(a + \tau)$ and centred in the origin that can be expressed with equation 19. B) A set of point charges (black circles) inside a sphere D_Q of radius a centred at $Q = (\tau, \alpha, \beta)$ creates a potential inside the sphere D_0 of radius a and centred in the origin that can be expressed with equation 21 (provided that $\tau > 2a$). In both A) and B), O represents the origin of coordinates, and $P = (r, \theta, \phi)$ is the point where the potential is measured. In our systems, the charges lie in equispaced grid points.

- Then, the box hierarchy is swept in the opposite direction: from lower to higher levels, the equation 21 is used to calculate the potential created by the boxes (the potential given by equation 21 is valid in regions that are not equal to those where equation 19 is valid).
- Finally, the total potential in every grid point ($v_{j,k,l}^{FMM}$) is calculated as an addition of three terms: the potentials due to nearby charges are directly calculated with the pairwise formula of equation 13, and the potentials due to the rest of the charges are calculated either with equation 19 or with equation 21, depending on the relative position of the boxes which create the potential and the box where the potential is evaluated.

In the whole procedure above, the charge in the grid point (j, k, l) is $\Omega\rho_{j,k,l}$. This scheme corresponds to the traditional version of FMM [43]. We used a slight modification of it [50] which not only converts multipoles between consecutive levels, but also within every given level, which enables further computational savings.

2.4 Conjugate gradients

In this section and in the following one we present two widely used iterative methods to calculate the electrostatic interaction: conjugate gradients and multigrid [51]. These methods are based on finding a solution to the Poisson equation 2 by starting from an initial guess and systematically refining it so that it becomes arbitrarily closer to the exact solution. These two methods have the advantage that if a good initial approximation is known, only a few iterations are required.

When using a grid representation, the Poisson equation can be discretised using finite differences. In this scheme the Laplacian at each grid point is approximated by a sum over the values of neighbouring points multiplied by certain weights. High-order expressions that include several neighbours can be used to control the error in the approximation [52]. The finite-difference approximation turns equation (2) into a linear algebraic equation

$$\tilde{L}\boldsymbol{x} = \boldsymbol{y} \ , \tag{23}$$

where $\tilde{L}$ is a sparse matrix (taking advantage of the sparsity of a system of equations can greatly reduce the numerical complexity of its solution [53]), $\boldsymbol{y}$ is a known vector ($y = -\rho/\epsilon_0$, in this case) and $\boldsymbol{x}$ is the vector we are solving for, in this case the electrostatic potential. Equation (23) can be efficiently solved by iterative methods that are based on the application of the matrix-vector product without the need to store the matrix.

Since the matrix is symmetric and positive definite, we can use the standard conjugate gradients [54] (CG) method. In the CG method, $\boldsymbol{x}$ is built as a linear combination of a set of orthogonal vectors $\boldsymbol{p}_k$. In every iteration, a new term is added

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \ , \tag{24}$$

which attempts to minimise

$$f(\boldsymbol{x}) := \frac{1}{2}\boldsymbol{x}^T\tilde{L}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{y} \ , \tag{25}$$

whose minimum is the solution of equation 23. The term added to the potential in iteration k is built so that $\boldsymbol{x}$ moves in the direction of the gradient of f, but being orthogonal to the previous terms. The gradient of $f(\boldsymbol{x})$ satisfies, $-\nabla f(\boldsymbol{x}) = \boldsymbol{y} - \tilde{L}\boldsymbol{x}$. Therefore the search direction in iteration $k+1$ is

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \frac{|\boldsymbol{r}_{k+1}|^2}{|\boldsymbol{r}_k|^2}\boldsymbol{p}_k \ . \tag{26}$$

with $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\tilde{L}\boldsymbol{p}_k$ and $\boldsymbol{p}_0 = \boldsymbol{r}_0 = \boldsymbol{y} - \tilde{L}\boldsymbol{x}_0$, being $\boldsymbol{x}_0$ arbitrary. The coefficient associated with each direction, α_k, is obtained from the minimization condition, yielding

$$\alpha_k = \frac{|\boldsymbol{r}_k|^2}{\boldsymbol{p}_k^T\tilde{L}\boldsymbol{p}_k} \ . \tag{27}$$

The equations above show that the CG method has a linear scaling ($\mathcal{O}(N)$) with the number of points N for a given number of iterations whenever the matrix $\tilde{L}$ has a number of non-zero entries per row that is much lower than N (as is the usual case for the Poisson equation). Bigger exponents in the scaling can appear, however, in problems where the number of performed iterations is chosen to keep the solution error below a given threshold that depends on N [55].

An issue that appears when using the finite-difference discretisation to solve the Poisson equation are boundary conditions: they must be given by setting the values of the points on the border of the domain. For free space boundary conditions we need a secondary method to obtain the value of the potential over these points; this additional method can represent a significant fraction of the computational cost and can introduce an approximation error. In our implementation we use a multipole expansion. Such an expansion concerns not only the boundaries: all the charges of the system are decomposed into two contributions: the first contribution is obtained with using a multipole expansion, and the corresponding potential is analytically calculated [28]; the potential created by the rest of the charge is calculated numerically with either the conjugate gradient method or the multigrid method. Splitting methods like this are commonly used in the literature [32].

2.5 Multigrid

Multigrid [56, 57, 58, 59, 60, 51] is a powerful method to solve elliptic partial differential equations, such as the Poisson problem [61], in an iterative fashion. Multigrid is routinely used as a solver or preconditioner for electronic structure and other scientific applications [62, 63, 64, 65]. In this Section we will make a brief introduction to a simple version of the multigrid approach that is adequate for the Poisson problem. Multigrid, however, can also be generalised to more complex problems, like non-linear problems [60] and systems where there is no direct geometric interpretation, in what is known as algebraic multigrid [66]. It has also been extended to solve eigenvalue problems [67, 64].

Multigrid is based on iterative solvers like Jacobi or Gauss-Seidel [51]. These methods are based on a simple iteration formula, that for equation 23 reads

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + M^{-1}(\boldsymbol{y} - \tilde{L}\boldsymbol{x}) \ . \tag{28}$$

The matrix M is an approximation to $\tilde{L}$ that is simple to invert. In the case of Jacobi, M is the diagonal of $\tilde{L}$, and for Gauss-Seidel, M is upper diagonal part of $\tilde{L}$. These methods are simple to implement, in particular in the case of the Laplacian operator, but are quite inefficient by themselves, so they are not practical as linear solvers for high-performance applications. However, they have a particular property: they are very good in removing the high-frequency error of a solution approximation, where the frequency is defined in relation to the grid spacing. In other words, given an approximation to the solution, a few iterations of Jacobi or Gauss-Seidel will make the solution smooth. In multigrid the smoothing property is exploited by using a hierarchy of grids of different spacing, and hence changing the frequency that these smoothing operators can remove efficiently.

A fundamental concept in multigrid is the residual of a linear equation. If we have $\bar{x}$ as an approximate solution of equation 23, the associated residual, $\boldsymbol{b}$, is defined as

$$\boldsymbol{b} = \boldsymbol{y} - \tilde{L}\bar{\boldsymbol{x}} \ . \tag{29}$$

We can use the residual to define an alternative linear problem

$$\tilde{L}\boldsymbol{a} = \boldsymbol{b} \ . \tag{30}$$

Due to the linearity of the Laplacian operator, finding $\boldsymbol{a}$ is equivalent to solving the original linear problem, equation 23, as

$$\boldsymbol{x} = \bar{\boldsymbol{x}} + \boldsymbol{a}. \tag{31}$$

If a few iterations of a smoothing operator have been applied to the approximate solution, $\bar{x}$, we know that the high-frequency components of the corresponding residual $\boldsymbol{b}$ should be small. Then it is possible to represent $\boldsymbol{b}$ in a grid that has, for example, two times the spacing without too much loss of accuracy. In this coarser grid equation 30 can be solved with less computational cost. Once the solution $\boldsymbol{a}$ is found on the coarser grid, it can be transferred back to the original grid and used to improve the approximation to the solution using equation 31.

The concept of calculating a correction term in a coarser grid is the basis of the multigrid approach. Instead of two grids, as in our previous example, a hierarchy of grids is used, at each level the residual is transferred to a coarser grid where the correction term is calculated. This is done up to the coarsest level that only contains a few points. Then the correction is calculated and transferred back to the finer levels.

To properly define the multigrid algorithm it is necessary to specify the operators that transfer functions between grids of different spacing. For transferring to a finer grid, typically a linear interpolation is used. For transferring to a coarser grid a so-called restriction operator is used. In a restriction operator, the value of the coarse grid point is calculated as a weighted average of the values of the corresponding points in the fine grid and its neighbours.

Now we introduce the multigrid algorithm in detail. Each quantity is labelled by a super-index that identifies the associated grid level, with 0 being the coarsest grid and L the finest. We denote S^l as the smoothing operator at level l, which corresponds to a few steps (usually 2 or 3) of Gauss-Seidel or Jacobi. I_l^m represents the transference of a function from the level l to the level m by restriction or interpolation. Following these conventions, we introduce the algorithm of a multigrid iteration in Figure 2. Given an initial approximation for the solution, we perform a few steps of the smoothing operator, after which the residual is calculated and transferred to the coarser grid. This iteration is repeated until the coarsest level is reached. Then we start to move towards finer grids. In each step the approximate solution of each level is interpolated into the finer grid and added, as a correction term, to the solution approximation of that level, after which a few steps of smoothing are performed. Finally, when the finest level is reached, a correction term that has contributions from the whole grid hierarchy is added to the initial approximation to the solution.

The scheme presented in Figure 2 is known as a v-cycle, for the order in which levels are visited, some more sophisticated strategies exist, where the coarsest level is visited twice (a w-cycle) or more times before coming back to the finest level [60]. Usually a v-cycle reduces the error, measured as the norm of the residual, by around one order of magnitude, so typically several v-cycles are required to find a converged solution.

When a good initial approximation is not known, an approach known as full multigrid (FMG) can be used. In FMG the original problem is solved first in the coarsest grid, then the solution is interpolated to the next grid in the hierarchy, where it is used as initial guess. The process is repeated until the finest grid is reached. It has been shown that the cost of solving the Poisson equation by FMG depends linearly with the number of grid points [60].

```
**Multigrid $\boldsymbol{v}$-cycle**
Input: $\boldsymbol{y}, \bar{\boldsymbol{x}}$
Output: $\bar{\boldsymbol{x}}$

$\boldsymbol{y}^L \leftarrow \boldsymbol{y}$
$\boldsymbol{x}^L \leftarrow \bar{\boldsymbol{x}}$
for $l$ from $L$ to 0 do
  if $l \neq L$ then
    $x^l \leftarrow 0$ {Set initial guess to 0}
  end if
  $\boldsymbol{x}^l \leftarrow \mathrm{S}^l \boldsymbol{x}^l$ {Pre-smoothing}
  if $l \neq 0$ then
    $\boldsymbol{b}^l \leftarrow \boldsymbol{y}^l - \tilde{L}^l \boldsymbol{x}^l$ {Calculate the residual}
    $\boldsymbol{y}^{l-1} \leftarrow I_l^{l-1} \boldsymbol{b}^l$ {Transfer the residual to the coarser grid}
  end if
end for
for $l$ from 0 to $L$ do
  if $l \neq 0$ then
    $\boldsymbol{x}^l \leftarrow \boldsymbol{x}^l + I_{l-1}^l \boldsymbol{x}^{l-1}$ {Transfer the correction to the finer grid}
  end if
  $\boldsymbol{x}^l \leftarrow \mathrm{S}^l \boldsymbol{x}^l$ {Post-smoothing}
end for
$\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}^L$
```

Figure 2: Algorithm of a multigrid v-cycle.

3 Methodology

3.1 Implementation

For our tests on the features of Poisson solvers in the context of quantum mechanics, we chose the OCTOPUS code [68, 69, 21], since it is representative of the trends for quantum ab initio simulation for the Poisson problem. OCTOPUS is a program for quantum *ab initio* simulations based on DFT and TDDFT for finite and periodic systems under the presence of arbitrary external static and time-dependent fields, which uses a real-space grid representation instead of using a basis of functions. This allows for a systematic control of the discretisation error, of particular importance for excited-state properties [70]. Furthermore, it uses a multi-level parallelisation scheme where the data is distributed following a tree-based approach. This scheme uses the message passing interface (MPI) library for coarse grain parallelisation and OpenMP for fine grain. This design has indeed been shown to be quite efficient in modern parallel computers [21].

Because the block of calculating the Hartree potential is made in a real space representation in most simulation packages, the results of our comparison are not particular to OCTOPUS and are rather general and package-independent. However, other implementations could show slight differences in their performance features. To ensure a fair comparison between the methods, we tried to use implementations as efficient as possible. In particular, state-of-the-art massively parallel implementations were used in the case of the FFT, ISF, and FMM methods. The corresponding packages are publicly available, and they can be integrated into other codes. The conjugate gradients and multigrid solvers are less portable, since they are *ad hoc* implementations for our code (they do not appear in isolated libraries). Although there exist more competitive implementations of conjugate gradients and multigrid, our tests of them are expected to provide conclusions that can be extrapolated.

Further details about each implementation are given next.

3.1.1 Parallel fast Fourier transform

In three dimensions a discrete FT of size $n_1 \times n_2 \times n_3$ can be evaluated using 1D FFTs along each direction, yielding a fast algorithm with arithmetic complexity $\mathcal{O}(n_1 n_2 n_3 \log_2(n_1 n_2 n_3))$. In addition, the one-dimensional decomposition of the 3D-FFT provides a straightforward parallelisation strategy based on a domain decomposition strategy. We now present the two-dimensional data decomposition that was first proposed by H. Q. Ding *et al.* [71] and later implemented by M. Eleftheriou *et al.* [72, 73, 74].

The starting point is to decompose the input data set along the first two dimensions into equal blocks of size $\frac{n_1}{P_1} \times \frac{n_2}{P_2} \times n_3$ and distribute these blocks on a two-dimensional grid of $P_1 \times P_2$ processes. Therefore, each process can perform $\frac{n_1}{P_1} \times \frac{n_2}{P_2}$ one-dimensional FFTs of size n_3 locally. Afterwards, a communication step is performed that redistributes the data along directions 1 and 3 in blocks of size $\frac{n_1}{P_1} \times n_2 \times \frac{n_3}{P_2}$, such that the 1D-FFT along direction 2 can be performed locally on each process. Then, a second communication step is performed, that redistributes the data along direction 2 and 3 in blocks of size $n_1 \times \frac{n_2}{P_1} \times \frac{n_3}{P_2}$. Now, the 3D-FFT is completed by performing the 1D-FFTs along direction 1. This algorithm is illustrated in Figure 3. For $n_1 \geq n_2 \geq n_3$ the two-dimensional data decomposition allows the usage of at most $n_2 \times n_3$ processes.

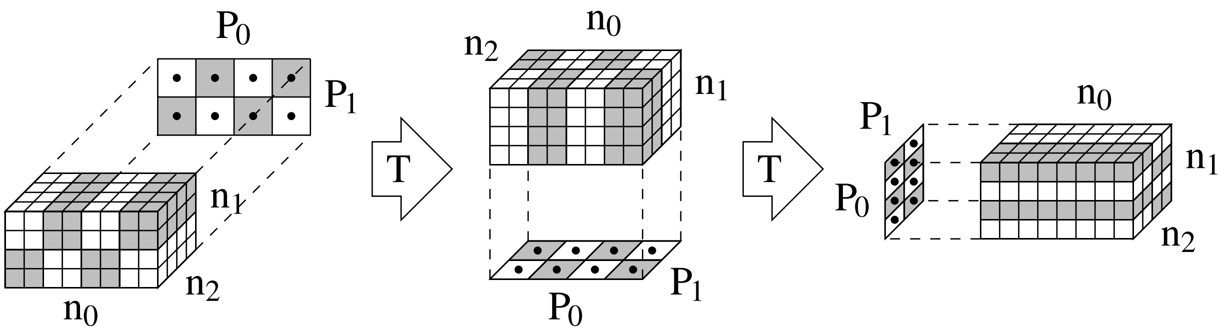


Figure 3: Distribution of a three-dimensional dataset of size $n_1 \times n_2 \times n_3 = 8 \times 4 \times 4$ on a two-dimensional process grid of size $P_1 \times P_2 = 4 \times 2$.

Since it is not trivial to implement an efficient FFT routine in parallel, we rely on an optimised implementation. Fortunately, there are several publicly available FFT software libraries based on the two-dimensional data decomposition. Among them are the FFT package from Sandia National Laboratories [75, 76], the P3DFFT software library [77, 78], the 2DECOMP&FFT package [79, 80], and the PFFT software library [81, 82]. Other efficient implementations exist, but unfortunately they are not distributed as stand-alone packages [83]. Our test runs are implemented with the help of the PFFT software library [81], which utilises the FFTW [84, 85] software package for the one-dimensional FFTs and the global communication steps. PFFT has a similar performance to the well-known P3DFFT, as well as some user-interface advantages [81], so we chose it for our survey.

3.1.2 Interpolating scaling functions

For the ISF solver we used the latest version of the original package provided by its authors [14]. This version is distributed as part of the BIGDFT code [37], but can be compiled as a standalone library, and we will therefore refer to it as libISF. This library includes its own parallel FFT routine [39], which divides the cell into smaller parallelepiped plane-like domains disjoints in its x (or y or z) coordinate, so we expect this part of the solver to have different scaling properties than the solver based on the PFFT library.

The accuracy of the ISF solver is defined by the choice of expanding the $1/r$ kernel into 81 Gaussians. Choosing a larger number of Gaussians would result in more precision, without any increase in the computations (only the initialisation will be slightly slower), yet the default value of 81 is appropriate for our purposes.

3.1.3 Fast multipole method

In this case we have used a massively parallel version of the fast multipole method [44], which we will denote as libFMM. The concrete implementation used in this work is explained in [50, 44] and it is included in the SCAFACOS package (“scalable fast Coulomb solver”)[86], which is a general parallel library for solving the Coulomb problem. Additionally to the libFMM library, it also provides other state-of-the-art methods with a common interface.

Effective parallelisation of this method is attained by a domain decomposition. Moreover, libFMM allows the user to tune the relative error of the calculations. Its expression is the quotient $(E_{ref} - E_n)/E_n$, where E_n is the Hartree energy calculated with the FMM method and E_{ref} is an estimation of what its actual value is. We chose for our calculations a relative error of 10^{-4}. Note that this error corresponds only to the pairwise term of the Hartree potential, before the correction for charge distribution is applied (see sections 2.3 and S7).

3.1.4 Conjugate gradients and multigrid

Different versions of the conjugate gradients and multigrid algorithms can have a big efficiency difference for some problems [87]. In this article, we chose the seminal version of conjugate gradients [54] and the standard version of multigrid using Gauss-Seidel smoothing (with 1 cycle for presmoothing and 4 cycles for postsmoothing). We choose them because they are at present widely used. The Hestenes version is among the most popular versions of conjugate gradients [88, 89, 90] and it is commonly used in programs and libraries seeking efficiency [91, 88]. The multigrid algorithm with Gauss-Seidel smoothing is also quite popular at present [92, 93, 87]. The fact that our implementations of multigrid and conjugate gradients might not be the fastest available is because of their accuracy issues: on the one hand, both methods fail when the shape of the simulation grid is not compact and the nuclei are not far from its border (which precludes most of the practical cases); on the other hand, even for such boxes, the accuracy of the calculations for both methods is much lower than that of the most competitive solvers (i.e. ISF, FMM, and PFFT), as will be shown in Section 4.1. The limited accuracy of multigrid and conjugate gradients for our problem did not make it advisable to focus on thorough implementations for them. Therefore, the efficiency properties of multigrid and conjugate gradients presented in Section 4.2 must be understood as an estimate of the properties of the corresponding solver families, in contrast to the efficiency properties of the most accurate methods (ISF, FMM, PFFT), whose implementations correspond to libraries thoroughly selected among the most efficient existing ones.

It is to be stressed that the multigrid and conjugate gradient solvers do not produce appropriate results when the set of grid points corresponds to an *irregular* shape (e.g., a grid shape that is the addition of spheres centred in each nucleus of the test system, which is the default case in OCTOPUS), and must be used with compact shapes, such as spherical or parallelepiped. This is because in irregular boxes we impose the value of the density to be strictly 0 in some points, and representing it with a smooth series as a multipole expansion can lead to very steep changes in the density, and therefore to errors.

With respect to the parallelisation of our CG implementation, it is based on the domain decomposition approach, where the grid is divided into subregions that are assigned to each process. Since the application of the finite-difference Laplacian only requires near-neighbour values, only the boundary values need to be shared between processors (see refs. [68, 21] for details). Since our implementation can work with arbitrary (irregular) shape grids, dividing the grids into subdomains of similar volume while minimising the area of the boundaries is not a trivial problem, so the Metis library [94] is used for this task.

Just as in the case of conjugate gradients, the parallelisation of multigrid is based on the domain decomposition approach. However in the case of multigrid some additional complications appear. First of all, as coarser grids are used the domain decomposition approach becomes less efficient as the number of points per domain is reduced. Secondly, the Gauss-Seidel procedure used for smoothing should be applied to each point sequentially [60] so it is not suitable for domain decomposition. In our implementation we take the simple approach of applying Gauss-Seidel in parallel over each domain. For a large number of domains, this scheme would in fact converge to the less-efficient Jacobi approach.

Concerning the chosen input parameters for our test calculations, we fix the multipole expansion (the order of the multipole expansion of the charges whose potential is analytically calculated [28]) to 7 for multigrid and conjugate gradients. In addition, for multigrid we have used all available multigrid levels (number of stages in the grid hierarchy of the multigrid solver). Further information about the concrete parameters can be found in the Section S4 of the supporting info.

3.2 Computational platforms

For carrying our tests out, we have used computational resources in Europe: Curie in France —at Commissariat à l'Energie Atomique (CEA)— and two IBM Blue Gene/P machines in Germany —Jugene at the Jülich Supercomputing Center, and Genius at the Rechenzentrum Garching of the Max Planck Society—. In addition, we have run some tests in a small cluster (Corvo) that represents parallel machines maintained locally by research institutions. All four machines are considered to be representative of the current trends of scientific High Performance Computing [95, 96, 97].

The IBM Blue Gene/P system is based on using a large number of low power processors. Each chip consists of 4 cores and it is associated to only 2 GiB of RAM. There are 294,912 cores in Jugene, 16,384 cores in Genius. The Blue Gene/P has a very sophisticated interconnection, each computing node of Blue Gene/P has 4 communication networks: (a) a 3D torus network for point-to-point communications, 2 connections per dimension; (b) a network for collective communications; (c) a network for barriers; and (d) a network for control. On the other hand, the Curie supercomputer is based on Intel Xeon processors. Each compute node has 32 cores (4 chips of 8 cores each) and 128 GiB of RAM. The compute nodes are connected with an Infiniband network. In total 11,520 cores and almost 46 TB of RAM are available. Finally, the configuration of the cluster Corvo is similar to the Curie supercomputer, although in a much lower scale. Each node of Corvo has two Intel Xeon with 6 cores each and 48 GiB of RAM. There are 960 cores in total, connected with Infiniband.

4 Results

In this Section we present the results of our tests to measure the execution time and accuracy of the methods discussed in Section 2. Although in principle the accuracy should not depend significantly on the particular implementation of the method used, the same is obviously not true for the execution time. Therefore, in order to avoid any ambiguity, whenever the term Poisson solver is used in the following, it always refers to a particular implementation of a given method. The results of some more tests and scalability comparisons are presented in the supporting information accompanying this paper.

Our tests consist of the calculation of the Hartree potential v for a given charge density ρ, but the efficiency conclusions would be also valid for cases where a classical generalised potential ($(v, \mathbf{A})$, analogous to the Hartree potential) is calculated as a function of the charge density and the magnetisation ($(\rho, \mathbf{m})$, or equivalently $(\rho_\uparrow, \rho_\downarrow)$). This is because such a "classical magnetic term" would be also pairwise.

4.1 Accuracy

All the quantities involved in *ab initio* calculations have to be calculated with as lowest possible errors to ensure the desired accuracy of the final result, and the Hartree potential is not an exception. In order to gauge the accuracy of the analysed solvers, we calculate the Hartree potential created by Gaussian charge distributions. Such potentials can be analytically calculated, and hence the error made by any method can be measured by comparing the two results: numerical and analytical. The chosen input charge distributions correspond to Gaussian functions $\rho_a(\boldsymbol{r}) = \exp(-|\boldsymbol{r}|^2/\alpha^2)/\alpha^3\pi^{3/2}$, with α chosen to be 16 times the spacing, i.e. 3.2 Å (this guarantees that the smoothness of the Gaussian test function is similar to the smoothness of densities of general physical problems). Such charge distributions are represented in cubic grids with variable size ($2L_e$) and constant spacing between consecutive points (spacing = 0.2 Å) in all three directions. The use of other values for the spacing does not alter the accuracy in a wide value range, and increases the numerical complexity proportionally with the corresponding increase of the grid points (see Figure S4 of the supporting information). We use two different quantities to measure the accuracy of a given method, the error in the potential, Ξ_v, and the error in the electrostatic energy, Ξ_E. We define them as

$$\Xi_v := \frac{\sum_{ijk} |v_a(\boldsymbol{r}_{ijk}) - v_n(\boldsymbol{r}_{ijk})|}{\sum_{ijk} |v_a(\boldsymbol{r}_{ijk})|} \ , \tag{32a}$$

$$E_a = \frac{1}{2} \int \mathrm{d}\boldsymbol{r} \ \rho_a(\boldsymbol{r}) v_a(\boldsymbol{r}) = -\frac{2}{\alpha^3 \pi^{1/2}} \int_0^\infty \mathrm{d}r \, r \, \exp(-r^2/\alpha^2) \, \mathrm{erf}(r/\alpha) \ , \tag{32b}$$

$$E_n = \frac{1}{2} \sum_{ijk} \rho(\boldsymbol{r}_{ijk}) v_n(\boldsymbol{r}_{ijk}) \ , \tag{32c}$$

$$\Xi_E := \frac{E_a - E_n}{E_a} \ , \tag{32d}$$

where $\boldsymbol{r}_{ijk}$ are all the points of the analysed grid, $r = |\boldsymbol{r}|$, and erf stands for the error function. v_a is the analytically-calculated Hartree potential, v_n is the potential calculated numerically, and E_a and E_n, respectively, are their associated electrostatic energies. These quantities provide an estimate of the deviations of the calculated potential and energy from their exact values. Ξ_v gives a comprehensive estimate of the error in the calculations of the Hartree potential, for it takes into account the calculated potential in all points with equal weight, while Ξ_E provides an estimate of the error in the potential in high-density regions.

In Table 1 we display the errors in the potential and the energy for the tested methods. The FFT and ISF methods provide in general best accuracies. Our implementation of the multigrid solver does not work for big grid sizes due to memory limitations. The errors in energy Ξ_E are less sensitive to local deviations than the errors in potential Ξ_v, since the former are weighted with the density. Ξ_E varies just lightly with the different sizes for the most reliable methods (ISF, FFT, and FMM, though FMM is less accurate than the other two). The values of Ξ_v are also essentially constant for ISF and FMM. However, for the FFT method an increase of the accuracy with the system size is clear. This is because the solution of the FFT corresponds to the superposition of infinite periodic images; for smaller sizes, the neighbour images are closer from the centre of the grid where we measure the potential, and therefore the images contribute more significantly to distort the potential (the closer the centres of two Gaussian functions, the bigger their superposition). The errors in both potential and energy of the multigrid method oscillate a bit, as a consequence of the different number of used multigrid levels. The errors for the conjugate gradients method increase with the size of the system because the bigger the number of points, the harder it is for the multipolar expansion (see Section 2.4) to adapt to all the grid points. As stated in Section 3.1, the multigrid and conjugate gradients methods do not provide acceptable accuracies whenever the simulation grid is not compact. The accuracy of the FMM method is mainly limited by the approximation of the charge densities as sets of discrete charges (the results obtained with the FMM are almost identical to those of direct pairwise summation).

Potential error, Ξ_v:

L_e (Å)	FFT	ISF	FMM	CG	Multigrid
7.0	$3{\cdot}10^{-4}$	$6{\cdot}10^{-9}$	$9{\cdot}10^{-5}$	$3{\cdot}10^{-5}$	$1{\cdot}10^{-6}$
10.0	$2{\cdot}10^{-8}$	$1{\cdot}10^{-9}$	$2{\cdot}10^{-4}$	$3{\cdot}10^{-5}$	$3{\cdot}10^{-7}$
15.8	$1{\cdot}10^{-8}$	$1{\cdot}10^{-9}$	$2{\cdot}10^{-4}$	$5{\cdot}10^{-5}$	$4{\cdot}10^{-6}$
22.0	$2{\cdot}10^{-10}$	$2{\cdot}10^{-9}$	$4{\cdot}10^{-4}$	$5{\cdot}10^{-4}$	$8{\cdot}10^{-7}$
25.8	$< 9 \cdot 10^{-13}$	$3{\cdot}10^{-9}$	$4{\cdot}10^{-4}$	$6{\cdot}10^{-3}$	—
31.6	$< 9 \cdot 10^{-13}$	$3{\cdot}10^{-9}$	$4{\cdot}10^{-4}$	$1{\cdot}10^{-2}$	—

Energy error, Ξ_E (eV):

L_e (Å)	FFT	ISF	FMM	CG	Multigrid
7.0	$2{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$5{\cdot}10^{-6}$	$2{\cdot}10^{-5}$	$1{\cdot}10^{-6}$
10.0	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$6{\cdot}10^{-6}$	$5{\cdot}10^{-6}$	$4{\cdot}10^{-7}$
15.8	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$6{\cdot}10^{-6}$	$5{\cdot}10^{-5}$	$2{\cdot}10^{-6}$
22.0	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$5{\cdot}10^{-6}$	$5{\cdot}10^{-4}$	$6{\cdot}10^{-7}$
25.8	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$7{\cdot}10^{-6}$	$3{\cdot}10^{-3}$	—
31.6	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$6{\cdot}10^{-6}$	$6{\cdot}10^{-3}$	—

Table 1: Potential, Ξ_v, and energy, Ξ_v, errors of different methods in the calculation of the Hartree potential created by a Gaussian charge distribution represented on cubic grids of variable edge $2L_e$ Å and spacing 0.2 Å.

In order to assess how the accuracy of the Poisson solver affects an actual DFT calculation, we calculated the ground state of a system of chlorophyll containing 180 atoms [98] (Figure S5 of the supporting information). To this end, we used pseudopotentials (Troullier and Martins type), so 460 electrons are treated in the calculation. The grid shape was a set of spheres of radius 4.0 Å centred at the nuclei, and the grid spacing was 0.23 Å. The exchange correlation (xc) functional used was the VWN-LDA functional [99]. In Table 2 we display the value of the Hartree energy, the highest eigenvalue, and the HOMO-LUMO gap[1]. The FFT method is expected to provide the most accurate results (by considering the results of Table 1). The maximum difference in the Hartree energy divided by the number of electrons is less than 0.015 eV. The maximum difference in the HOMO-LUMO gap is less than 0.0092 eV. In this test case, the differences among all five methods can be considered negligible (much lower than the errors introduced by the XC functional, the pseudopotentials and the discretisation). However, note that the deviation of the Hartree energy for the multigrid and conjugate gradients methods with respect to the most accurate method (FFT) is rather larger than that of the ISF method, and so are the HOMO and the HOMO-LUMO gap.

	Hartree energy (eV)	HOMO (eV)	HOMO-LUMO gap (eV)
FFT	240,820.70	-4.8922	1.4471
ISF	240,821.48	-4.8935	1.4489
FMM	240,817.29	-4.8906	1.4429
CG	240,815.01	-4.9009	1.4521
Multigrid	240,814.69	-4.8979	1.4506

Table 2: Influence of the different methods on the Hartree energy, HOMO energy level and HOMO-LUMO gap corresponding to the ground state (calculated through DFT with pseudopotentials) of a chlorophyll stretch with 180 atoms.

4.2 Efficiency

In order to gauge the performance of the Poisson solvers, we have measured the solution time for each solver as a function of the number of processes. This measured time only includes the operations directly related to the solution of the Poisson equation and excludes the initialisation time of the Poisson solver. We ran one MPI process per node on Blue Gene/P's due to limited amount of memory per node[2] and one single MPI process per core on Curie and Corvo. In the latter two, further tests of the efficiency of OpenMP were also done[3]. Runs up to 4096 MPI processes were made in Genius, Jugene (both BlueGene/P), and Curie (x86-64 architecture) machines. Runs up to 512 were made in Corvo (also x86-64).

In our efficiency tests we calculated the potential created by Gaussian charge distributions as those explained in Section 4.1. These charge distributions are represented in parallelepiped grids with edge length $2L_e$, for L_e equal to 7.0, 10.0, 15.8, 22.0, 25.8 and 31.6 Å respectively. The grid points were equispaced with a spacing of 0.2 Å (additional tests with variable spacing are presented in Section S4 of the supporting information). The smallest simulated

[1]The HOMO-LUMO gap is often used to test the accuracy of a calculation method [100] because it provides an estimate for the first electronic excitation energy [101]. The HOMO itself provides an estimate for the ionization energy [102].

[2]Definitions of basic concepts on high performance computing (e.g. "node" and "core") can be found in [17].

[3]Multithreading is implemented in PFFT and libISF, our conjugate gradients solver shows only a small improvement, and the libFMM and multigrid solvers do not take advantage of this feature.

system, with $L_e = 7.0$ Å, contained 357,911 grid points, while the largest one with $L_e = 31.6$ Å contained 31,855,013 grid points. Due to memory constraints, the serial FFT was limited to $L_e = 15.8$ Å (4,019,679 grid points), multigrid to $L_e = 22.0$ Å (10,793,861 grid points) and the parallel runs for all solvers in the Blue Gene/P were limited up to $L_e = 25.8$ Å (17,373,979 points). As explained in Section 2.1, the FFT method requires an enlarged cell when applied to finite systems. The size of this grid ranges from 143^3 to 637^3 points (up to 525^3 on a Blue Gene/P's).

As stated in Section 3.1, the efficiency properties of our implementation of the multigrid and conjugate gradients solvers presented in this section must be understood as an estimate of the properties of the corresponding methods, in contrast to the efficiency properties of the most accurate methods (ISF, FMM, and FFT), which correspond to highly optimized (state-of-the-art) algorithms and implementations (libISF, libFMM, and PFFT).

Figure 4 presents the execution times obtained in tests on a Blue Gene/P machine, each graph representing a different problem size. Figure 5 presents the execution times in x86-64 machines and Figure 6 the comparison among both architectures. Weak-scaling tests are summarised in Figure 7 and OpenMP ones in Figure 8. These graphs can help to establish conclusions on the relative efficiency of the studied Poisson solvers. For deeper insight, tests comparing cubic and non-cubic grids with the same number of points are shown in Figure 9. In addition, speed-up comparisons, computational complexities fits, spacing analysis and multipolar corrections execution times are displayed in the supporting information.

As can be observed in Figure 4, the tests show a similar trend with regard to the system size and the number of MPI processes: execution times decrease with the number of processes until saturation, and larger systems allow the efficient use of a higher number of parallel processes, i.e., the solvers "saturates" at a higher number of processes. This is especially true for the PFFT and libFMM solvers. Thus, this behaviour leaves the way open to simulate physical systems of more realistic size if tens of thousands of cores are available.

For a given system size (N points), libISF is the fastest solver if the number of processes used in the solution is low-medium. One of the reasons for this behaviour is that, in contrast to the FFT method, libISF does not require the box size to be increased for accuracy reasons. This advantage should nevertheless disappear when dealing with periodic systems. The execution time of libISF decreases with the number of MPI processes until a saturation point (P_m processes) is reached. P_m, the congestion point, is proportional to N_{cx} (or N_{cy} or N_{cz}) being $N_{cx} \times N_{cy} \times N_{cz}$ the number of points of the minimal parallelepiped grid containing the original grid (such an auxiliary grid is necessary for the discrete Fourier transform which is carried out by libISF). This is because libISF divides such a parallelepiped into smaller parallelepiped plane-like domains disjoints in its x (or y or z) coordinate. Since the minimal width of one of these domains is 1 (one point in direction x), the maximum number of MPI processes to use in a parallel run is roughly N_{cx} (actually the minimal execution times commonly happens for a number of MPI processes which is slightly bigger –e.g. 10 % – than N_{cx}; this is because the Goedecker's FFT used by ISF works only for particular values of grid sizes, so the grid must often be slightly enlarged to match a doable size). Using more MPI processes leads to increased execution times, because all the processes take part in the internal `MPI_Alltoallv` communications, even if they have not done any useful work. This behaviour can be delayed using OpenMP threads inside MPI processes. As we will see later in Figure 8, the best efficiency is achieved with a combination of MPI processes (that

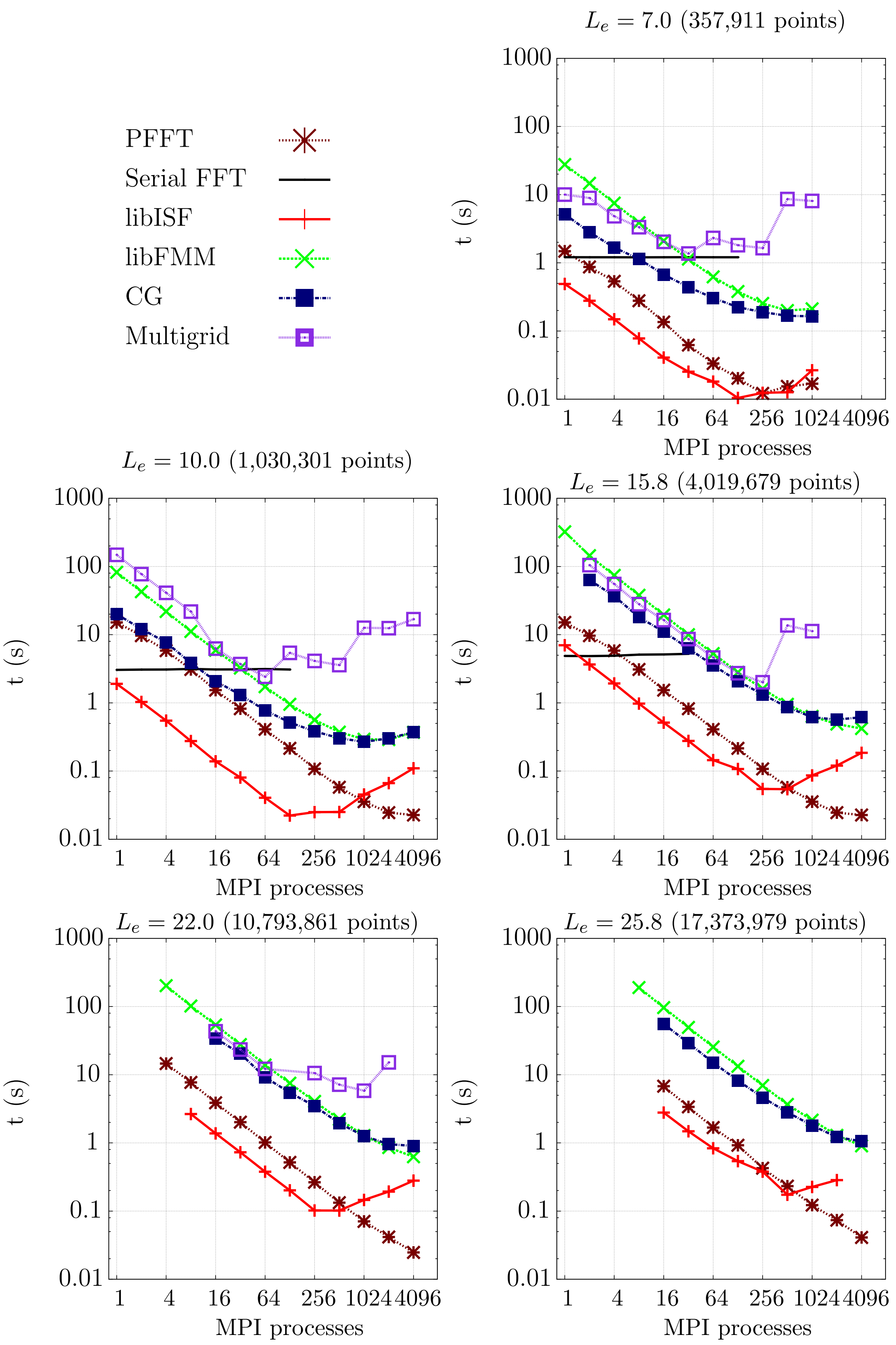


Figure 4: Execution times for the calculation of the Hartree potential created by a Gaussian charge distribution on a Blue Gene/P machine as a function of the number of MPI processes for six different Poisson solvers and for different simulated system sizes (number of grid points).

are limited by the grid size) and OpenMP threads, thus using a higher number of cores in parallel.

The conjugate gradient and libFMM solvers seem to be very efficient approaches in terms of scalability. Particularly attractive is the good performance obtained with the PFFT solver. If the number of processes is high enough, the execution times are always lower than the corresponding to other solvers. A better approach to the data distribution and highly effective communications are the reasons behind. Furthermore, the prefactor of the FFT method is small by construction, and this largely compensates the fact that FFT does scale with $N \log_2 N$ (so, higher than N). Nevertheless, when the number of processes is low enough, PFFT is not as fast as libISF, since the FFT method requires the box size to be increased, as explained above.

The execution times obtained using the multigrid solver depend on the chosen number of multigrid levels, being faster when the number of levels is high. The number of levels in the Multigrid solver is equal to $\lfloor \log_8 N \rfloor - 3$, where N is the total amount of grid points. However, for a fixed problem size, the number of levels in practice must be reduced as the number of processes increases, due to the need to assure a minimum workload to each processor. Results (see Figures 4 and 5) show that the multigrid solver offers the poorest performance even when using a high number of levels. Every processor needs a minimum number of points to work with; this implies a maximum number of usable multigrid levels for a given number of processors (and this number may be lower than $\lfloor \log_8 N \rfloor - 3$). When this *optimal* number of multigrid levels is not reachable (e.g., for a high number of processors), then the convergence of the multigrid solver is slower, and the execution time increases (see Figure 4). This saturation point appears with a relatively low number of parallel processes.

The multigrid and conjugate gradients methods require the calculation of a correction due to the multipolar expansion used. The calculation of this correction implies a similar time for both our solvers, and represents between 0.2% and 8% of the conjugate gradients solver total execution time, and between 1.7% and 3.2% in the case of the multigrid solver. In essence, it does not add a significant extra time. Further details can be found in Section S6 of the supporting information.

Our tests have shown that the implementations of the novel Poisson solvers, PFFT and libFMM, do offer good scalability and accuracy, and could be used efficiently when hundreds or thousands of parallel processes are needed for the analysis of representative systems. libISF should be the chosen solver when low-medium numbers of processes are used, and PFFT should be chosen instead for high numbers of processes (the concrete number of processes that makes PFFT more competitive depends on the system size). Almost linear performances can be observed until saturation for all solvers. Before saturation, the obtained efficiency factors are always above 50%. As expected, large systems, which have higher computation needs, can make better use of a high number of processes.

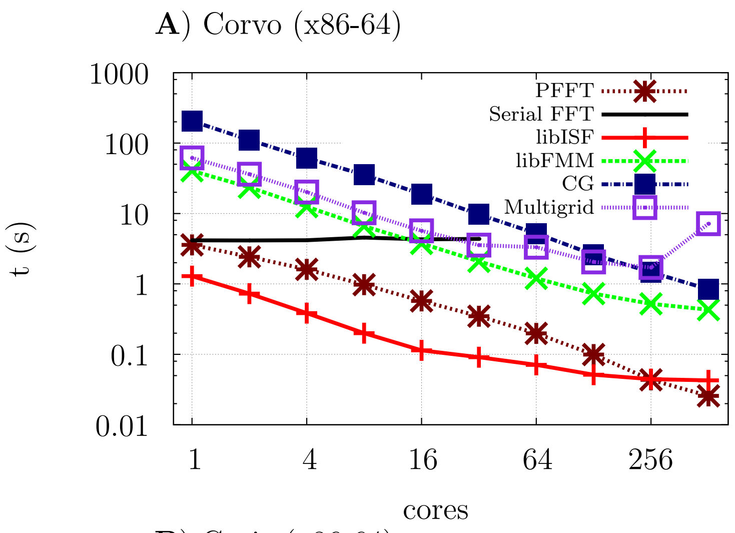
A similar overall behaviour of the different solvers, with minor differences, is also shown in the other two machines, Curie and Corvo. Nevertheless, one appreciable difference is that the networks are not as efficient as in Blue Gene/P, and when a solver saturates, the execution time increases substantially. Figure 5 presents the obtained results for a particular case: $L_e = 15.8$ Å (4,019,679 grid points). As it can be observed, the best results are obtained again with the PFFT solver for a high number of parallel processes and with libISF for a lower-medium number of processes (the libISF solver needs to combine OpenMP and MPI approaches, otherwise the time increases heavily when using a high number of processes). Relative performance between the solvers varies slightly from the above analysed cases, but the conclusions are very similar.

For the serial case, executions are between 1.4 and 5 times faster in Curie than in Corvo for the different solvers and system sizes, but this difference tends to disappear in the parallel versions, where communication times must be added to computing time. These differences can be attributed to the different cache sizes and abilities for vector processing of the processors of Curie (X7560-Nehalem) and Corvo (E5645-Westmere).

In order to compare more clearly the results obtained with different computers, Figure 6 shows the execution times of a concrete solver (PFFT) for the case of $L_e = 15.8$ Å (4,019,679 grid points) in Genius (Blue Gene/P architecture), Curie (x86-64) and Corvo (x86-64). Execution times are higher in the Blue Gene/P computer when the number of processes is relatively small, but it allows an efficient use of a high number of processes before saturation. The processors of the Blue Gene/P machine are slower than those of Curie and Corvo, but communication infrastructures are more effective. This leads us to conclude that the execution time of the Poisson solver parallel code depends not only on the individual processor performance, but also on interprocessor communication network of the parallel computer, specially when using a high number of processors.

Apart from the execution time of a concrete parallel run, a very useful quantity to measure in order to estimate its performance is the weak-scaling. Weak-scaling measures the ability of a program to scale with the number of parallel processes at a fixed computing load, comparing the execution time of a problem of size N using P processes with the execution time of the analogous problem of size kN using kP processes. In this way, each process will use the same amount of computational resources, regardless of the number of used processors. If the program is mainly constricted by computational needs, then the execution times should be very similar in both cases. On the other hand, if communication needs dominate the parallel execution, then the execution times will grow with P, usually faster than linearly. In order to maintain the same number of grid points per processor in all cases, we executed different system sizes and, for these tests, adapted them to L_e equal to 7.8 Å (493,039 grid points), 10.0 Å (1,030,301 points), 15.8 Å (4,019,679 points), 20.0 Å (8,120,601 points), 25.0 Å (15,813,251 points) and 31.6 Å (31,855,013 points, only in Corvo) for the Poisson solver on Blue Gene/P and Corvo. We did parallel runs using 4, 8, 32, 64 and 128 MPI processes (also 216 and 256 processes in Corvo). So, the number of grid points processed in each parallel MPI process is roughly 125,000 for all the cases. Figure 7 shows the obtained results, with data taken from the profiling output. In it, the normalised weak-scaling for P processes is measured as the quotient: [time to calculate the Hartree potential of a system of about 125,000$\times P$ points using P processors] divided by [time to calculate the Hartree potential of a system of 493,039 points with 4 processors].

The weak-scaling factor of the conjugate gradient and multigrid solvers shows an important increase with the number of processes, and this can be used to predict that parallel execution will saturate at a moderate number of processes (as observed in Figure 4). In contrast, libFMM shows the best results in the analysed range: weak-scaling increases very moderately, so we can conclude that communication times are acceptable and that the solver will offer good speed-up results when using thousands of processes. PFFT also gives very good results in the Blue Gene/P system; they are similar to those of libFMM, so similar



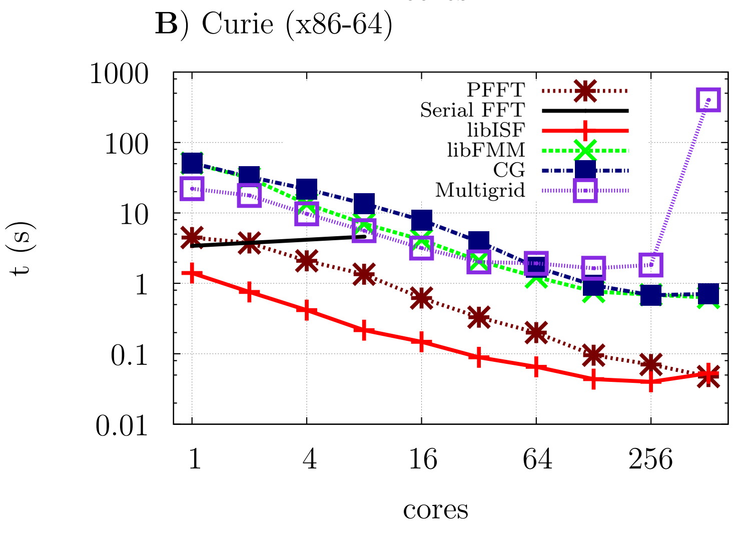


Figure 5: Execution times of the Poisson solvers (L_e = 15.8 Å - 4,019,679 grid points) in Corvo (A) and Curie (B) supercomputers as a function of the number of MPI processes (each one executed in a CPU processor core). The execution times for the libISF solver correspond to the fastest MPI+OpenMP combination (see Figure 8 for more details).

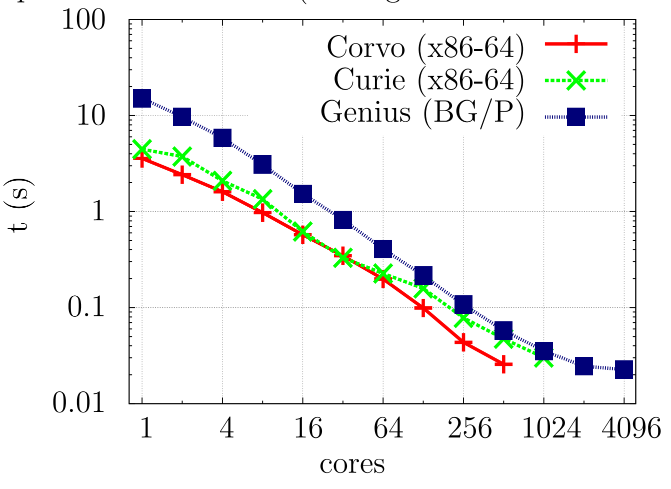
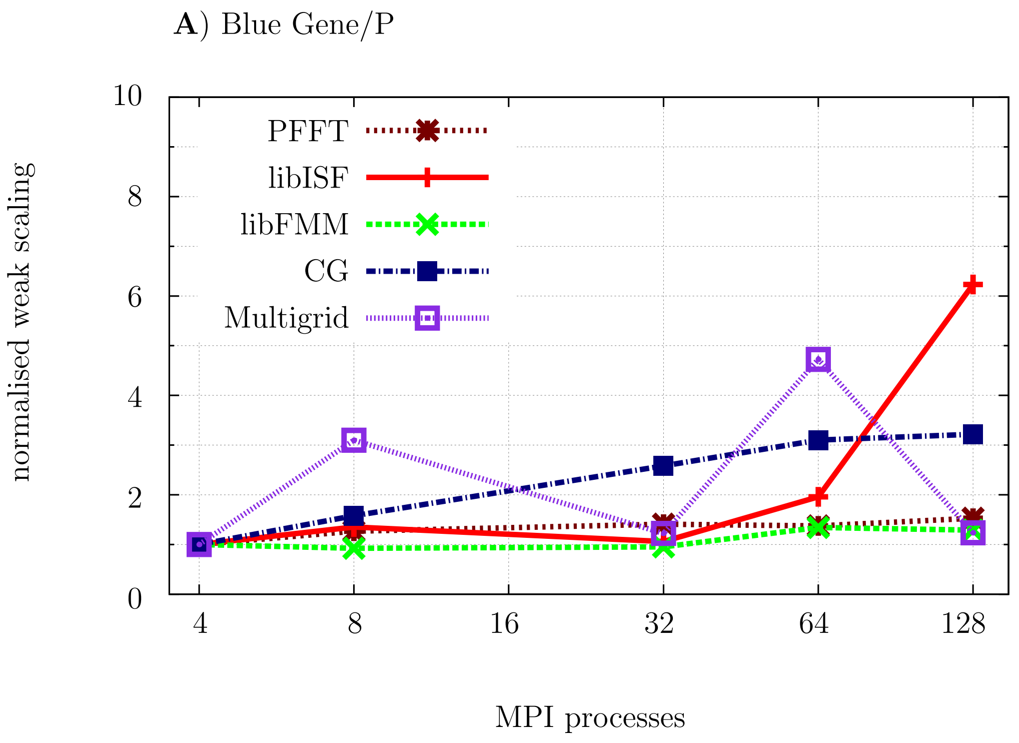


Figure 6: Execution times of the PFFT solver in Genius (Blue Gene/P architecture), Curie and Corvo (x86-64) for a system size of $L_e = 15.8$ Å (4,019,679 grid points) as a function of the number of MPI processes.



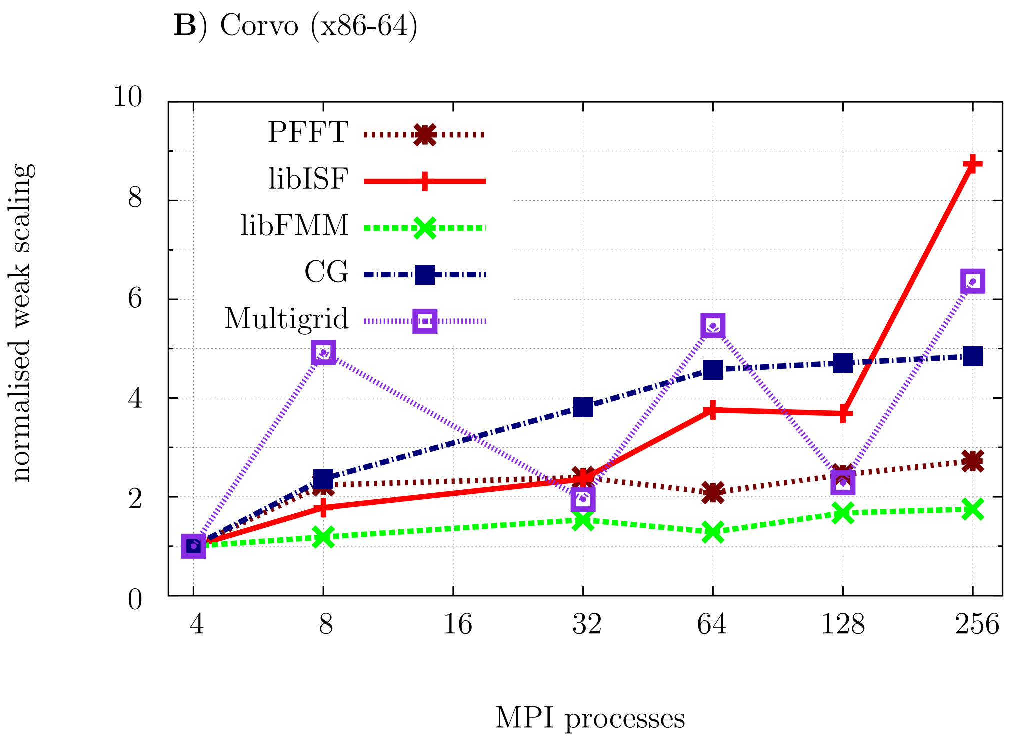
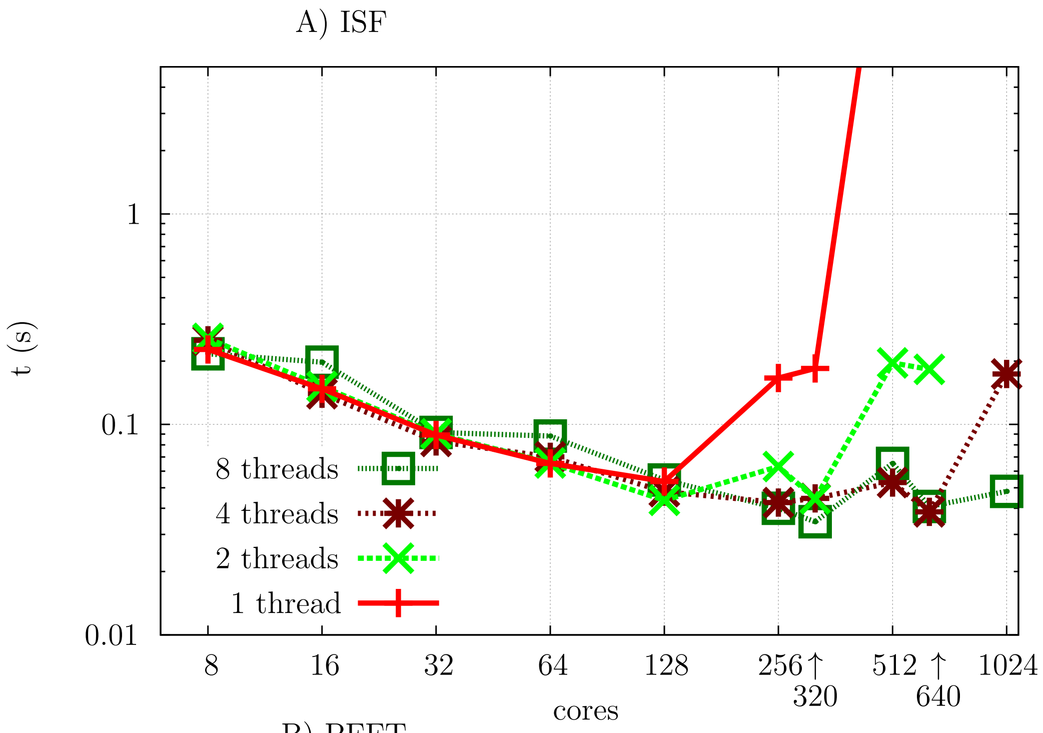


Figure 7: Normalised weak-scaling (i.e. execution time for the calculation of the Hartree potential in a system of about 125,000×P points using P processors divided by the execution time for a system of 493,039 points with 4 processors) of the Poisson solvers in Blue Gene/P (A) and Corvo (x86-64) (B). Each MPI process has roughly 125,000 points and selected system sizes are given by L_e equal to 7.8 Å (493,039 grid points), 10.0 Å (1,030,301 points), 15.8 Å (4,019,679 points), 20.0 Å (8,120,601 points), 25.0 Å (15,813,251 points) and 31.6 Å (31,855,013 points, only in Corvo) and a spacing of 0.2 Å.

conclusions can be stated. Nonetheless, the results obtained in Corvo up to 256 processes seem to indicate a trend to increase, which implies that communications will play a more important role if a high number of processes is to be used. Similar behaviour is shown with the libISF solver, but with a sharper increase with large number of processes. Thus, Figure 7 highlights the efficiency of the communication network and protocols of the parallel computer: the weak-scaling factor is markedly better and bounded in Blue Gene/P, a system with a very high performance communication network. From these tests, we can conclude that the communication needs of the Poisson solvers fit better in the network of a Blue Gene/P machine than in that of a machine with x86-64 processors and Infiniband network (e.g. Corvo). This stresses the importance of efficient communication architectures when dealing with the calculation of pairwise potentials.

Multithreading tests with OpenMP were also done in order to gauge the maximum performance for all cores. The analysed libFMM, conjugate gradients and multigrid solvers do not reduce significantly the execution times when increasing the number of OpenMP threads for a given number of MPI processes, while the PFFT and libISF solvers do show such a reduction (see Section S3 of the supporting information). Figure 8 shows the execution times in Corvo (x86-64) for the system of $L_e = 15.8$ Å and spacing 0.2 Å as a function of the number of cores and the number of OpenMP threads for the libISF (A) and the PFFT (B) solvers. For each number of cores, we have measured the execution times varying the number of OpenMP threads per MPI process (maintaining always the number of parallel tasks equal to the total number of cores). For the case of the PFFT solver, the best results are always obtained when all the available cores are used for MPI processes (without OpenMP threads). Conversely, for the libISF solver the combination between OpenMP and MPI should be chosen with more care; for a low number of cores (less than the number of grid planes), it seems to be better to use a MPI process per core (with very small differences compared with other combinations). Instead, if a higher number of cores is available, then OpenMP threads must be used to overcome the performance limits imposed by the number of grid planes, so keeping the number of MPI process below the number of planes. In any case, the maximum number of OpenMP threads that can be used is limited by the actual machine, in this case between 4 (Blue Gene/P machines) and 32 (Curie x86-64).

A final test was done using a non-cubic box: the Poisson solver timings of a parallelepiped grid of size 15.8 Å $\times$ 15.8 Å $\times$ 32.0 Å (solid lines in Figure 9) were compared with those of a cubic grid with $L_e = 20.0$ Å(dashed lines in Figure 9). Both grids had a spacing of 0.2 Å and the number of grid points were as similar as possible (8,115,201 and 8,120,601 grid points). As it can be viewed in Figure 9, the execution times of the libISF solver are quite similar for both grids. The conjugate gradients and multigrid solvers show only small differences between both cases, although the execution times are slightly lower in the non-cubic grid. Conversely, libFMM times are around 50% higher in the non-cubic grid. Above all, the PFFT solver shows the biggest penalty here, with execution times being 75% higher in the non-cubic grid. The reason behind this is that, for finite systems, the FFT method that we use requires the use of an enlarged grid of cubic shape (see Sec. 2.1). The penalty should disappear when solving the Poisson equation for periodic systems, as the enlargement of the grid is not required in those cases. This penalty is expected to make the crossover between libISF and PFFT to occur at higher values of the number of cores for systems contained



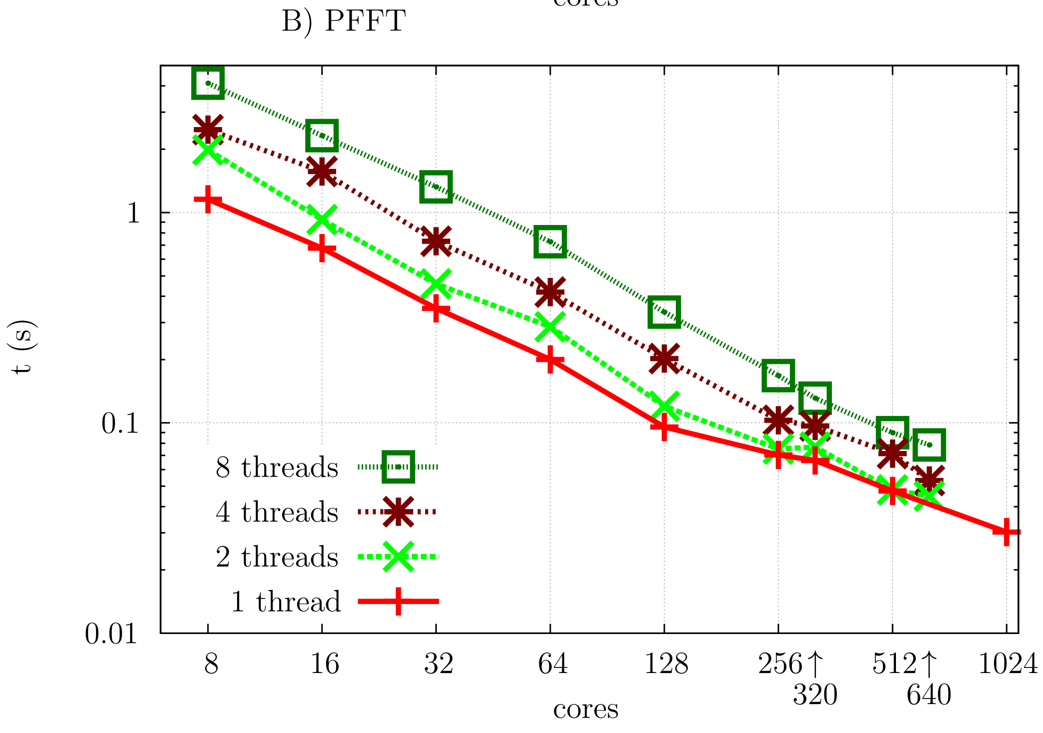


Figure 8: Execution times for the system of $L_e = 15.8$ Å (4,019,679 grid points) corresponding to the A) libISF solver and to the B) PFFT solver, using from 8 to 1024 cores, and varying the number of OpenMP threads from 1 to 8, in Curie (x86-64). The product of [number of MPI processes] $\times$ [number of OpenMP threads] is always equal to the number of used cores. The same trend is also shown in the Corvo (x86-64) machine.

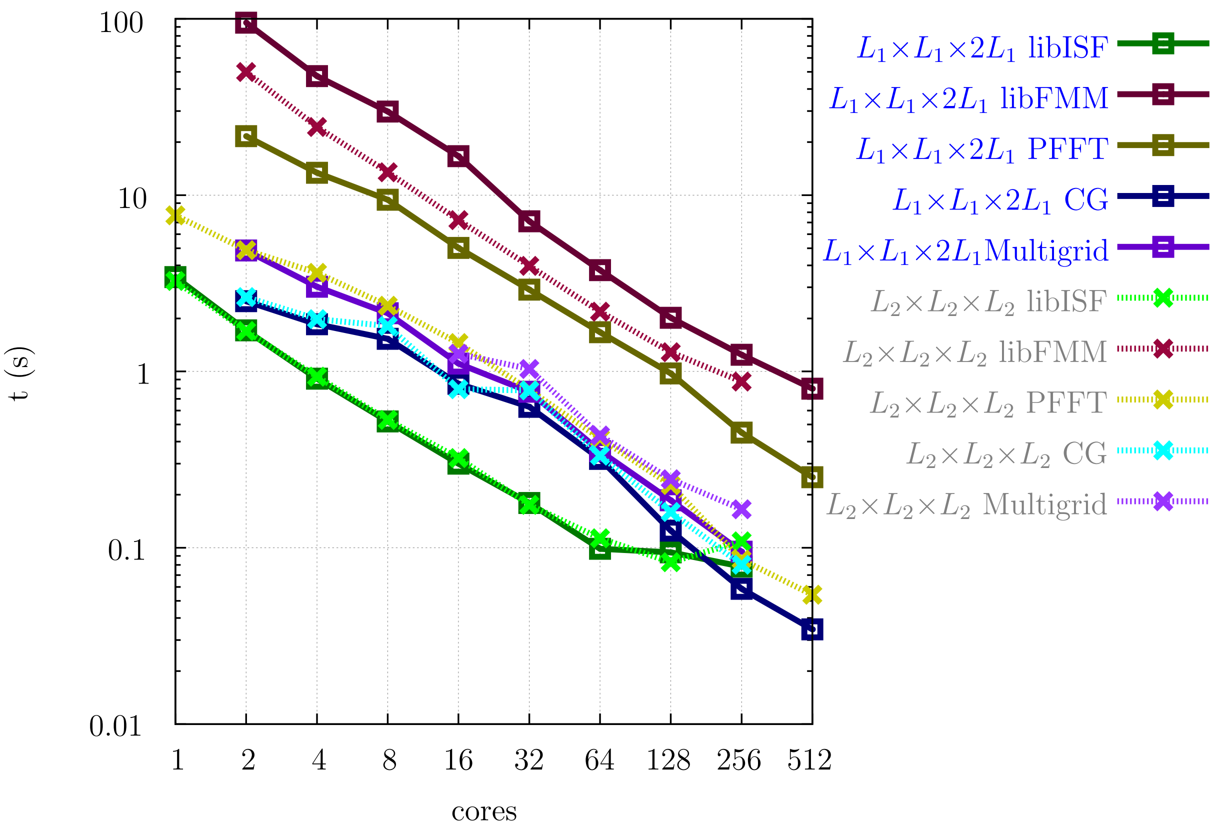


Figure 9: Execution times for the Hartree potential calculation as a function of the solver for a grid of size 15.8 Å × 15.8 Å × 32.0 Å(solid line) and for a grid of size 20.0 Å × 20.0 Å × 20.0 Å(dashed line), in Corvo (x86-64) with a unique OpenMP thread per MPI process. Spacing equal 0.2 Å; both grids contain approximately the same number of points.

in non-cubic grids[4] (if compared with cubig grid with the same number of points). For a given number of grid points $N = N_1 \times N_2 \times N_3$ (assuming $N_1 \geq N_2, N_3$), the bigger N_1 in comparison with N_2, N_3, the bigger the expected number of cores for the crossover.

For very time-consuming simulations, some tests should be performed in order to choose the optimal Poisson solver. In addition, for every given simulation code, it would be advisable to implement one subroutine which automatically chooses the solver which is expected to be optimal for the tackled problem.

5 Conclusions

In this paper we analysed the relative performance of several implementations of celebrated methods (fast Fourier transform, interpolating scaling functions, fast multipole method, conjugate gradients and multigrid) for the calculation of the classical Hartree potential created

[4]Please note that despite the fact that grids for DFT and TDDFT calculations are usually irregular, both libISF and PFFT require the application of discrete Fourier transforms, and therefore a cubic grid containing the original grid is used for the calculation of the Hartree potential when using libISF or PFFT.

by charge distributions represented in real space. The first part of the paper presents the fundamentals of these calculation methods and their corresponding parallel implementations. In the second part, we summarise the computational aspects of the tests we carried out to measure the implementations' relative performances. These tests were run on three kinds of supercomputers, which were chosen as representative of present-day high performance facilities. In our tests, we focused on measuring accuracies, execution times, speed-ups, and weak-scaling parameters. Test runs involved up to 4,096 parallel processes, and solved system sizes from about 350,000 grid points to about 32,000,000 grid points.

Our results show that the PFFT solver is the most efficient option for a high number of cores, so PFFT should be the default option to calculate the Hartree potential when using a number of cores which is beyond a given (problem dependent) threshold. For lower number of cores, the libISF solver should be preferred. We traced back this different behaviour to the parallelisation strategy of the FFT used in both cases. Indeed, the two-dimensional data decomposition of PFFT goes beyond the one-dimensional data decomposition used in the libISF package. One can point that, for the sake of enhanced efficiency of the Poisson solver, the future implementation of an ISF method which makes use of the PFFT library would be of great interest. The specific number of cores which make PFFT faster than ISF depends on the given problem: the grid shape, the number of grid points, and also the computing facility. In addition, it will also depend on the MPI/OpenMP balance chosen for the ISF solver. For a given number of grid points, the bigger the deviation of the grid shape from a cube, the bigger the number of cores for the ISF-PFFT efficiency crossover.

In some special cases, the charges might not lie in equispaced points (e.g. when curvilinear coordinates are used), and so the PFFT and libISF methods cannot be used, as they require the calculation of FFTs. In these cases, the FMM should be chosen instead, since it works, it is linearly-scaling and it is accurate regardless of the charge density's spatial location. The libFMM solver also shows good performance and scaling, yet its accuracy is lower and its execution times are less competitive than those of the PFFT and libISF solvers on the analysed machines. In contrast to libISF, libFMM scales almost linearly up to high values of the number of processes, but since its numerical complexity has a larger prefactor, it would only be competitive with libISF when the number of parallel processes increases significantly. The performance of the conjugate gradients solver has a trend similar to that of libFMM, as does the multigrid solver for low values of the number of processes. Weak-scaling tests show that communication costs are the smallest for the libFMM solver, while they are acceptable for all the others.

ISF and FFT methods are more accurate than FMM, conjugate gradients, and multigrid methods. Hence, they should be chosen if accurate calculations of electrostatics are required. However, according to our tests, the accuracy of all the analysed methods is expected to be appropriate for density functional theory and time-dependent density functional theory calculations (where the calculation of the Hartree potential is an essential step) because these have other sources of error that will typically have a much stronger impact (see Section 4.1). Nevertheless, neither the multigrid nor the conjugate gradients methods can reach acceptable accuracy if the data set is not represented on a compact (spherical or parallelepiped) grid.

Acknowledgement

We would like to express our gratitude to José Luis Alonso, David Cardamone, Ivo Kabadshow, Miguel Marques and Ibai Gurrutxaga for support and illuminating advice. We acknowledge the PRACE Research Infrastructure resource Jugene based on Germany at Forschungszentrum Jülich and Curie based in France at CEA, funded by GENCI and the Rechenzentrum Garching (RZG) of the Max Planck Society for the usage of IBM Blue Gene/P. We specially acknowledge Laurent Nguyen, Serge Haddad and the Curie supercomputer management committee for their support, as well as Heiko Appel for the help given with the Genius supercomputer in Garching. We also acknowledge financial support from 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), general funding for research groups UPV/EHU (ALDAPA, GIU10/02), Grupos Consolidados UPV/EHU del Gobierno Vasco (IT-319-07) and European Commission project CRONOS (280879-2 CRONOS CP-FP7). P. García-Risueño is funded by the Humboldt Universität zu Berlin. J. Alberdi-Rodriguez acknowledges the scholarship of the University of the Basque Country UPV/EHU. M. Pippig was supported by the German Ministry of Science and Education (BMBF) under grant 01IH08001. M. J. T. Oliveira thankfully acknowledges financial support from the Portuguese FCT (contract #SFRH/BPD/44608/2008).

References

[1] P. Hohenberg and W. Kohn, *Inhomogeneous Electron Gas*, Phys. Rev. **136** (1964) B864–B871.

[2] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev. **140** (1965) A1133–A1138.

[3] E. Runge and E. K. U. Gross, *Density-functional theory for time-dependent systems*, Phys. Rev. Lett. **52** (1984) 997–1000.

[4] J. P. Perdew and A. Zunger, *Self-interaction correction to density-functional approximations for many-electron systems*, Phys. Rev. B **23** (1981) 5048–5079.

[5] N. Umezawa, *Explicit density-functional exchange potential with correct asymptotic behavior*, Phys. Rev. A **74** (2006) 032505.

[6] X. Andrade and A. Aspuru-Guzik, *Prediction of the Derivative Discontinuity in Density Functional Theory from an Electrostatic Description of the Exchange and Correlation Potential*, Phys. Rev. Lett. **107** (2011) 183002.

[7] D. R. Hartree, *The calculation of atomic structures*, J. Wiley, New York, 1957.

[8] V. A. Fock, *The Theory of space, time and gravitation*, Pergamon Press, 1964.

[9] M. E. Casida, C. Jamorski, F. Bohr, J. Guan, and D. R. Salahub, *Optical Properties from Density-Functional Theory*, chapter 9, pp. 145–163, American Chemical Society, 1996.

[10] K. R., A. E., and N. J., *A generalized fast multipole approach for Hartree-Fock and density functional computations*, Chemical Physics Letters **238** (1995) 173-179.

[11] T. L. Beck, *Real-space multigrid solution of electrostatics problems and the Kohn–Sham equations*, Int. J. Quant. Chem. **65** (1997) 477–486.

[12] C. A. Rozzi, D. Varsano, A. Marini, E. K. U. Gross, and A. Rubio, *Exact Coulomb cutoff technique for supercell calculations*, Phys. Rev. B **73** (2006) 205119.

[13] E. Rudberg and P. Salek, *Efficient implementation of the fast multipole method*, J. Chem. Phys. **125** (2006) 084106.

[14] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, and G. Beylkin, *Efficient solution of Poisson's equation with free boundary conditions*, J. Chem. Phys. **125** (2006) 074105, Genovese, L. and Deutsch, T. and Neelov, A. and Goedecker, S. and Beylkin, G.

[15] L. Genovese, T. Deutsch, and S. Goedecker, *Efficient and accurate three-dimensional Poisson solver for surface problems*, J. Chem. Phys. **127** (2007) 054704.

[16] A. Cerioni, L. Genovese, A. Mirone, and V. A. Sole, *Efficient and accurate solver of the three-dimensional screened and unscreened Poisson's equation with generic boundary conditions*, J. Chem. Phys. **137** (2012) 134108.

[17] P. García-Risueño and P. E. Ibáñez, *A review of High Performance Computing foundations for scientists*, International Journal of Modern Physics C **23** (2012) 1230001.

[18] F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber, and J. Lorenz, *Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform*, in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006, ACM.

[19] J. L. Alonso, X. Andrade, P. Echenique, F. Falceto, D. Prada-Gracia, and A. Rubio, *Efficient formalism for large-scale ab initio molecular dynamics based on time-dependent density functional theory*, Phys. Rev. Lett. **101** (2008) 096403.

[20] X. Andrade, A. Castro, D. Zueco, J. L. Alonso, P. Echenique, F. Falceto, and A. Rubio, *Modified Ehrenfest formalism for efficient large-scale ab initio molecular dynamics*, J. Chem. Theory Comput. **5** (2009) 728–742.

[21] X. Andrade, J. Alberdi-Rodriguez, D. A. Strubbe, M. J. T. Oliveira, F. Nogueira, A. Castro, J. Muguerza, A. Arruabarrena, S. G. Louie, A. Aspuru-Guzik, A. Rubio, and M. A. L. Marques, *Time-dependent density-functional theory in massively parallel computer architectures: the octopus project*, Journal of Physics: Condensed Matter **24** (2012) 233202.

[22] J. Alberdi-Rodriguez, *Analysis of performance and scaling of the scientific code Octopus*, LAP LAMBERT Academic Publishing, 2010.

[23] G. M. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, in *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pp. 483–485, New York, NY, USA, 1967, ACM.

[24] B. Fang, Y. Deng, and G. Martyna, *Performance of the 3D {FFT} on the 6D network torus {QCDOC} parallel supercomputer*, Computer Physics Communications **176** (2007) 531 - 538.

[25] A. R. Leach, *Molecular modelling: Principles and applications*, Pearson Education - Prentice Hall, Harlow, 2nd edition, 2001.

[26] J. Tomasi, B. Mennucci, and R. Cammi, *Quantum Mechanical Continuum Solvation Models*, Chem. Rev. **105** (2005) 2999–3094.

[27] M. H. Nayfeh and M. K. Brussel, *Electricity and magnetism*, John Wiley & sons, 1985.

[28] A. Castro, A. Rubio, and M. J. Stott, *Solution of Poisson's equation for finite systems using plane-wave methods*, Canadian Journal of Physics **81** (2003) 1151–1164.

[29] R. Olivares-Amaya, M. Stopa, X. Andrade, M. A. Watson, and A. Aspuru-Guzik, *Anion Stabilization in Electrostatic Environments*, The Journal of Physical Chemistry Letters **2** (2011) 682–688.

[30] M. A. Watson, D. Rappoport, E. M. Y. Lee, R. Olivares-Amaya, and A. Aspuru-Guzik, *Electronic structure calculations in arbitrary electrostatic environments*, The Journal of Chemical Physics **136** (2012) 024101.

[31] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comput. **19** (1965) 297–301.

[32] G. J. Martyna and M. E. Tuckerman, *A reciprocal space based method for treating long range interactions in ab initio and force-field-based calculations in cluster*, J. Chem. Phys. **110** (1999) 2810.

[33] T. Darden, D. York, and L. Pedersen, *Particle mesh Ewald: An Nlog(N) method for Ewald sums in large systems*, J. Chem. Phys. **98** (1993) 10089.

[34] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, Institute of Physics Publishing Ltd., Bristol, 1988.

[35] C. Van Loan, *Computational frameworks for the fast Fourier transform*, volume 10, Society for Industrial Mathematics, 1992.

[36] L. I. Bluestein, *A Linear Filtering Approach to the Computation of the Discrete Fourier Transform*, IEEE Trans AU **18** (1970) 451–455.

[37] L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, S. A. Ghasemi, A. Willand, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman, and R. Schneider, *Daubechies wavelets as a basis set for density functional pseudopotential calculations*, The Journal of Chemical Physics **129** (2008) 014109.

[38] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, PA, 1998.

[39] S. Goedecker, *Wavelets and their application for the solution of partial differential equations in physics*, Presses Polytechniques et Universitaires Romandes, 1998.

[40] L. F. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys. **73** (1987) 325–348.

[41] L. F. Greengard and V. Rokhlin, *The Rapid Evaluation of Potential Fields in Three Dimensions*, Springer Press, Berlin, Heidelberg, 1988.

[42] H. Cheng, L. F. Greengard, and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, J. Comp. Phys **155** (1999) 468–498.

[43] L. F. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numerica **6** (1997) 229.

[44] H. Dachsel, *An error controlled fast multipole method*, J. Chem. Phys. **132** (2010) 119901.

[45] B. A. Cipra, *The Best of the 20th Century: Editors Name Top 10 Algorithms*, SIAM news **33, 4** (2000) .

[46] C. A. White, B. G. Johnson, P. M. W. Gill, and M. Head-Gordon, *The continuous fast multipole method*, Chem. Phys. Lett. **230** (1994) 8–16.

[47] C. A. White, B. G. Johnson, P. M. W. Gill, and M. Head-Gordon, *Linear scaling density functional calculations via the continuous fast multipole method*, Chem. Phys. Lett. **253** (1996) 268–278.

[48] M. Strain, G. Scuseria, and M. Frisch, *Achieving Linear Scaling for the Electronic Quantum Coulomb Problem*, Science **271** (1996) 51–53.

[49] C. A. White and M. Head-Gordon, *Derivation and efficient implementation of the fast multipole method*, J. Chem. Phys. **101** (1994) 6593–6605.

[50] I. Kabadshow and H. Dachsel, *The Error-Controlled Fast Multipole Method for Open and Periodic Boundary Conditions*, in *Fast Methods for Long-Range Interactions in Complex Systems*, IAS Series, Volume 6, Forschungszentrum Jülich, Germany, 2010, CECAM.

[51] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[52] J. R. Chelikowsky, N. Troullier, and Y. Saad, *Finite-difference-pseudopotential method: Electronic structure calculations without a basis*, Phys. Rev. Lett. **72** (1994) 1240–1243.

[53] P. García-Risueño and P. Echenique, *Linearly scaling direct method for accurately inverting sparse banded matrices*, J. of Phys. A: Math. and Theor. **45** (2012) 065204.

[54] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. of research of the national bureau of standards **49** (1952) 409–436.

[55] M. V. Fernández-Serra, E. Artacho, and J. M. Soler, *Model Hessian for accelerating first-principles structure optimizations*, Phys. Rev. B **67** (2003) 100101.

[56] P. Wesseling, *An introduction to multigrid methods*, John Wiley & Sons, 1992.

[57] J. Zhang, *Fast and High Accuracy Multigrid Solution of the Three Dimensional Poisson Equation*, J. Comp. Phys. **149** (1998) 449–461.

[58] W. L. Briggs, *A multigrid tutorial*, Wiley, New York, 1987.

[59] A. Brandt, *Multi-Level Adaptive Solution to Boundary-Value Problems*, Math. Comput **31** (1977) 333–390.

[60] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001.

[61] C. Rostgaard, K. W. Jacobsen, and K. S. Thygesen, *Fully self-consistent GW calculations for molecules*, Phys. Rev. B **81** (2010) 085103.

[62] E. L. Briggs, D. J. Sullivan, and J. Bernholc, *Real-space multigrid-based approach to large-scale electronic structure calculations*, Phys. Rev. B **54** (1996) 14362–14375.

[63] T. T. Beck, *Real-space mesh techniques in density-functional theory*, Rev. Mod. Phys. **72** (2000) 1041-1080.

[64] T. Torsti, M. Heiskanen, M. J. Puska, and R. M. Nieminen, *MIKA: Multigrid-based program package for electronic structure calculations*, Int. J. Quant. Chem. **91** (2003) 171–176.

[65] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, *Real-space grid implementation of the projector augmented wave method*, Phys. Rev. B **71** (2005) 035109.

[66] Y. Shapira, *Matrix-Based Multigrid: Theory and Applications*, Numerical Methods and Algorithms, Kluwer Academic Publishers, 2003.

[67] J. Mandel and S. McCormick, *A multilevel variational method for $Au = \lambda Bu$ on composite grids*, J. Comput. Phys. **80** (1989) 442–452.

[68] A. Castro, H. Appel, M. Oliveira, C. Rozzi, X. Andrade, F. Lorenzen, M. Marques, E. Gross, and A. Rubio, *Octopus: a tool for the application of time-dependent density functional theory*, Phys. Stat. Sol. B **243** (2006) 2465–2488, The OCTOPUS code is available in a public Subversion repository `http://www.tddft.org/svn/octopus/trunk`.

[69] M. A. L. Marques, A. Castro, G. F. Bertsch, and A. Rubio, *octopus: a first-principles tool for excited electron-ion dynamics*, Computer Physics Communications **151** (2003) 60.

[70] F. D. Vila, D. A. Strubbe, Y. Takimoto, X. Andrade, A. Rubio, S. G. Louie, and J. J. Rehr, *Basis set effects on the hyperpolarizability of $CHCl_3$: Gaussian-type orbitals, numerical basis sets and real-space grids*, J. Chem. Phys. **133** (2010) 034111.

[71] H. Q. Ding, D. B. Gennery, and R. D. Ferraro, *A Portable 3D FFT Package for Distributed-Memory Parallel Architectures*, in *Proceedings of 7th SIAM Conference on Parallel Processing*, pp. 70–71, SIAM Press, 1995.

[72] M. Eleftheriou, J. E. Moreira, B. G. Fitch, and R. S. Germain, *A Volumetric FFT for BlueGene/L*, in *HiPC*, edited by T. M. Pinkston and V. K. Prasanna, volume 2913 of *Lecture Notes in Computer Science*, pp. 194–203, Springer, 2003.

[73] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain, *Scalable framework for 3D FFTs on the Blue Gene/L Supercomputer: Implementation and early performance measurements*, IBM Journal of Research and Development **49** (2005) 457–464.

[74] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain, *Performance Measurements of the 3D FFT on the Blue Gene/L Supercomputer*, in *Euro-Par 2005 Parallel Processing*, edited by J. C. Cunha and P. D. Medeiros, volume 3648 of *Lecture Notes in Computer Science*, pp. 795–803, Springer, 2005.

[75] S. J. Plimpton, R. Pollock, and M. Stevens, *Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations*, in *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, 1997)*, Philadelphia, 1997, SIAM.

[76] S. Plimpton, *Parallel FFT subroutine library*, 2011, http://www.sandia.gov/ sjplimp/docs/fft/README.html.

[77] D. Pekurovsky, *P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions*, SIAM J. Sci. Comput. **34** (2012) C192 – C209.

[78] D. Pekurovsky, *P3DFFT, Parallel FFT subroutine library*, 2011, http://www.sdsc.edu/us/resources/p3dfft.

[79] N. Li and S. Laizet, *2DECOMP & FFT - A Highly Scalable 2D Decomposition Library and FFT Interface*, in *Cray User Group 2010 conference*, pp. 1 – 13, Edinburgh, Scotland, 2010.

[80] N. Li, *2DECOMP&FFT, Parallel FFT subroutine library*, 2011, http://www.hector.ac.uk/cse/distributedcse/reports/incompact3d/incompact3d/index.html.

[81] M. Pippig, *PFFT - An Extension of FFTW to Massively Parallel Architectures*, SIAM J. Sci. Comput. **35** (2013) C213 – C236.

[82] M. Pippig, *PFFT, Parallel FFT subroutine library*, 2011.

[83] T. V. Truong Duy and T. Ozaki, *A three-dimensional domain decomposition method for large-scale DFT electronic structure calculations*, ArXiv e-prints (2012) .

[84] M. Frigo and S. G. Johnson, *The Design and Implementation of FFTW3*, Proceedings of the IEEE **93** (2005) 216–231.

[85] M. Frigo and S. G. Johnson, *FFTW, C subroutine library*, `http://www.fftw.org`, 2009.

[86] M. Bolten, F. Fahrenberger, R. Halver, F. Heber, M. Hofmann, I. Kabadshow, O. Lenz, M. Pippig, and G. Sutmann, *ScaFaCoS, C subroutine library*, `http://scafacos.github.com/`, 2013.

[87] J. Wang and R. Luo, *Assessment of linear finite-difference Poisson–Boltzmann solvers*, Journal of Computational Chemistry **31** (2010) 1689–1698.

[88] Y. Saad, J. Chelikowsky, and S. Shontz, *Numerical Methods for Electronic Structure Calculations of Materials*, SIAM Review **52** (2010) 3-54.

[89] K.-A. Mardal and R. Winther, *Preconditioning discretizations of systems of partial differential equations*, Numerical Linear Algebra with Applications **18** (2011) 1–40.

[90] V. Weijo, M. Randrianarivony, H. Harbrecht, and L. Frediani, *Wavelet formulation of the polarizable continuum model*, Journal of Computational Chemistry **31** (2010) 1469–1477.

[91] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang, *PETSc Users Manual, Revision 3.3*, Mathematics and Computer Science Division, Argonne National Laboratory, 2012.

[92] S. Takacs and W. Zulehner, *Convergence analysis of multigrid methods with collective point smoothers for optimal control problems*, Computing and Visualization in Science **14** (2011) 131-141.

[93] M. Adams, *A Distributed Memory Unstructured Gauss-Seidel Algorithm for Multigrid Smoothers*, in *Supercomputing, ACM/IEEE 2001 Conference*, pp. 14–14, 2001.

[94] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing **20** (1998) 359–392.

[95] G. D. Fletcher, D. G. Fedorov, S. R. Pruitt, T. L. Windus, and M. S. Gordon, *Large-Scale MP2 Calculations on the Blue Gene Architecture Using the Fragment Molecular Orbital Method*, Journal of Chemical Theory and Computation **8** (2012) 75–79.

[96] W. Jiang, M. Hodoscek, and B. Roux, *Computation of Absolute Hydration and Binding Free Energy with Free Energy Perturbation Distributed Replica-Exchange Molecular Dynamics*, Journal of Chemical Theory and Computation **5** (2009) 2583–2588.

[97] G. Houzeaux, R. de la Cruz, H. Owen, and M. Vázquez, *Parallel uniform mesh multiplication applied to a Navier–Stokes solver*, Computers and Fluids (2012) .

[98] Z. Liu, H. Yan, K. Wang, T. Kuang, J. Zhang, L. Gui, X. An, and W. Chang, *Crystal structure of spinach major light-harvesting complex at 2.72 angstrom resolution*, Nature **428** (2004) 287–292.

[99] S. H. Vosko, L. Wilk, and M. Nusair, *Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis*, Canadian Journal of Physics **58** (1980) 1200-1211.

[100] M. Wanko, P. García-Risueño, and A. Rubio, *Excited states of the green fluorescent protein chromophore: Performance of ab initio and semi-empirical methods*, physica status solidi (b) **249** (2012) 392–400.

[101] C.-G. Zhan, J. A. Nichols, and D. A. Dixon, *Ionization Potential, Electron Affinity, Electronegativity, Hardness, and Electron Excitation Energy: Molecular Properties from Density Functional Theory Orbital Energies*, The Journal of Physical Chemistry A **107** (2003) 4184-4195.

[102] J. F. Janak, *Proof that $\frac{\partial E}{\partial n_i} = \epsilon$ in density-functional theory*, Phys. Rev. B **18** (1978) 7165–7168.

Supporting information of *A survey of the parallel performance and the accuracy of Poisson solvers for electronic structure calculations*

Pablo García-Risueño*[1,2,3], Joseba Alberdi-Rodriguez[4,5], Micael J. T. Oliveira[6], Xavier Andrade[7], Michael Pippig[8], Javier Muguerza[4], Agustin Arruabarrena[4], and Angel Rubio [5,9]

[1]Institut für Physik, Humboldt Universität zu Berlin, Zum grossen Windkanal 6, 12489 Berlin, Germany
[2]Institute for Biocomputation and Physics of Complex Systems BIFI, Universidad de Zaragoza C/ Mariano Esquillor, 50018 Zaragoza (Spain)
[3]Instituto de Química Física Rocasolano (CSIC), C/ Serrano 119, 28006 Madrid, Spain
[4]Dept. of Computer Architecture and Technology, University of the Basque Country UPV/EHU, M. Lardizabal, 1, 20018 Donostia/San Sebastián, Spain
[5]Nano-Bio Spectroscopy Group and European Theoretical Spectroscopy Facility, Spanish node, University of the Basque Country UPV/EHU, Edif. Joxe Mari Korta, Av. Tolosa 72, 20018 Donostia/San Sebastián, Spain
[6]Center for Computational Physics, University of Coimbra, Rua Larga, 3004-516 Coimbra, Portugal
[7]Dept. of Chemistry and Chemical Biology, Harvard University, 12 Oxford street, Cambridge, MA 02138, USA
[8]Dept. of Mathematics, Chemnitz University of Technology, 09107 Chemnitz, Germany
[9] Centro de Física de Materiales, CSIC-UPV/EHU-MPC and DIPC, 20018 Donostia/San Sebastián, Spain

October 20, 2013

*risueno@physik.hu-berlin.de

Abstract

In this document we include some information to complement our paper. First, we add some remarks on the efficiency of the algorithms, showing their speed-ups in Section S1, and the usage of multithreading in Section S2. Then, we provide some statements on the communication patterns of PFFT [1], ISF [2], and FMM [3] used by our implementations in Section S3. Afterwards, we give more details on the tests we have done (Section S4). We also present the structure of the molecule we used for some accuracy tests (Section S5). We also compare the time of the multipolar expansion correction of the multigrid and conjugate gradients methods with the corresponding total execution time in Section S6. Finally, in Section S7, we discuss the correction term we devised to adapt the FMM method (originally devised to deal with point charges) to charge distributions.

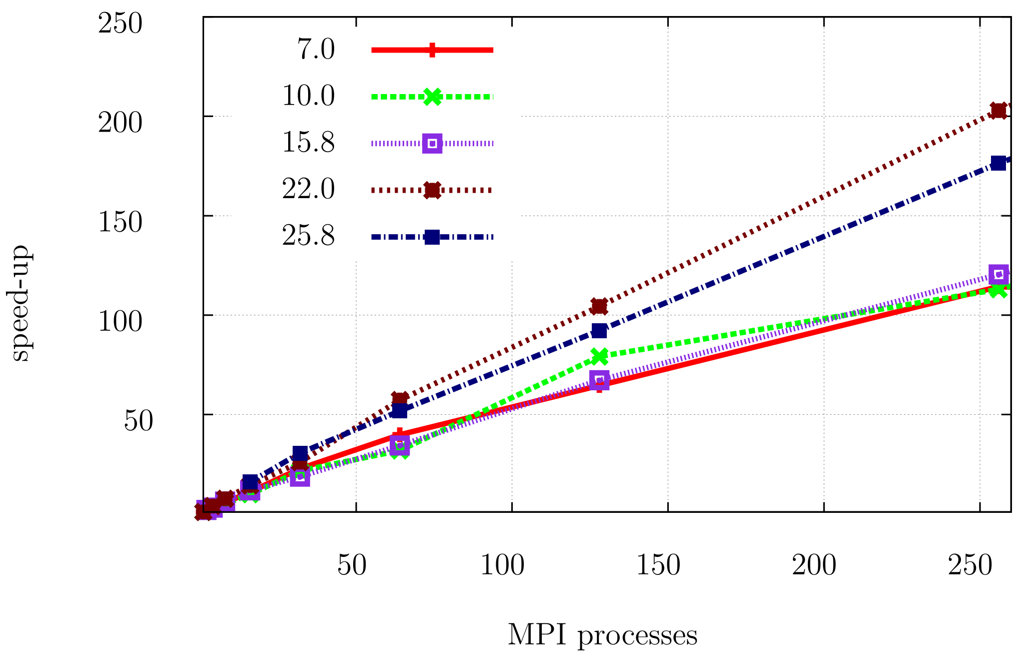
Contents

S1 Speed-ups

Our tests showed that the novel implementations of PFFT [1] and FMM [3] offer good scalability and accuracy, and are competitive if thousands of parallel processes are available. Figure S1 shows the speed-ups obtained using PFFT for the different system sizes in a BlueGene/P supercomputer. Almost linear performances can be observed until saturation for all the cases, and the obtained efficiencies have been always above 50% for just nearly all these points. As expected, large systems, which have higher computation needs, can make a better use of a high number of processes. Tests run in Corvo and Curie machines show similar trends (although with efficiency problems of PFFT for some values of MPI proc.).

We have also analysed the execution complexity by measuring the execution time on Blue Gene/P as a function of the volume of the system for PFFT and FMM in three cases: 16, 32 and 64 parallel processes. The grid points ratio among the smallest ($L_e = 7.0$ Å) and

A)



B)

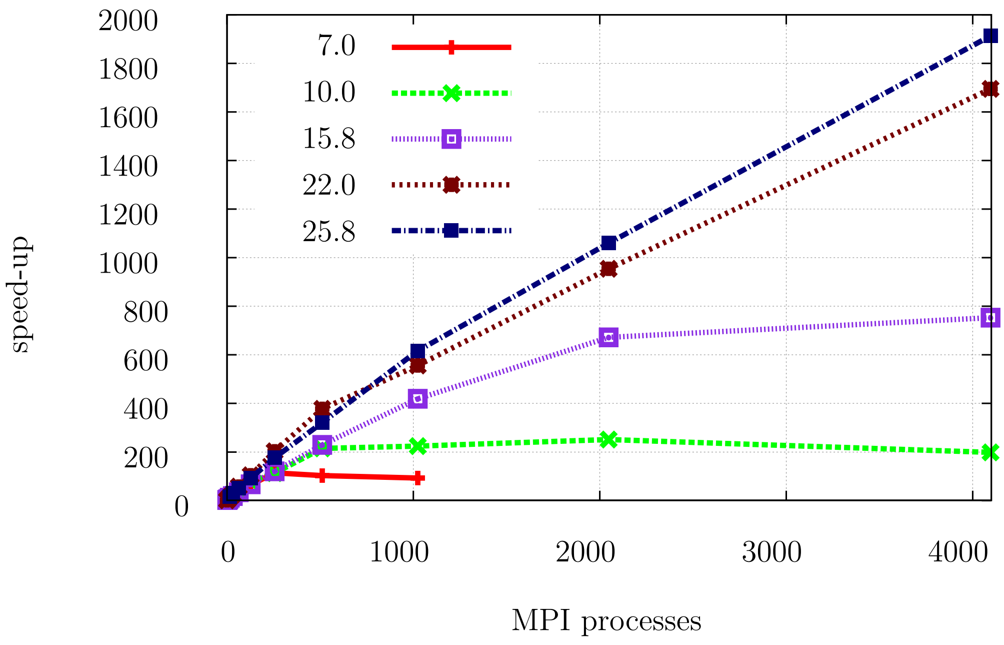


Figure S1: Speed-up of the PFFT Poisson solver in a Blue Gene/P computer for different system sizes (given by L_e, semi-legth of the parallelepiped edge). Largest systems saturate with more processes than the smallers ones. A) linear speed-up is shown up to 256 processes, independently of the simulated system. B) saturation point is higher when the simulated system is larger.

the largest ($L_e = 25.8$ Å) studied systems is about 49. Our results agree with the $\mathcal{O}(N)$ theoretical complexity of the FMM method and the $\mathcal{O}(N \log N)$ theoretical complexity of the PFFT method (see Figure S2). Note that computation time is appreciably higher when using the FMM solver due to its higher prefactor (i.e. the quantity that multiplies N or $N \log N$ in the expression of the numerical complexity). The quotient between prefactors is machine dependent; its value is about 5.5 for Corvo, about 9 for Curie and about 12 for Blue Gene/P. One may think that for very big values of N at a constant number of processes P, the growth of the term $\log N$ would eventually make FMM faster than PFFT. This is strictly true, but the extremely huge N of this eventual crossover precludes it in practice.

S2 Multithreading

Runs using a unique MPI process with many different number of OpenMP threads were done for all the analysed solvers. The tests summarised in Figure S3 correspond to runs of the cubic system of edge $L_e = 15.8$ Å and spacing 0.2 Å, using a Gaussian charge distribution, with all the solvers in a unique node of the Corvo computer (x86-64 architecture), and with a unique MPI process in all the cases. One can see no improvement for multigrid and FMM solvers using OpenMP, while for the conjugate gradients a very slight improvement appears with 2 OpenMP threads. In contrast, ISF and PFFT are faster whilst more threads are used. We can say that both show a very similar trend, although the increase of the performance of the ISF library is slightly higher than that of PFFT.

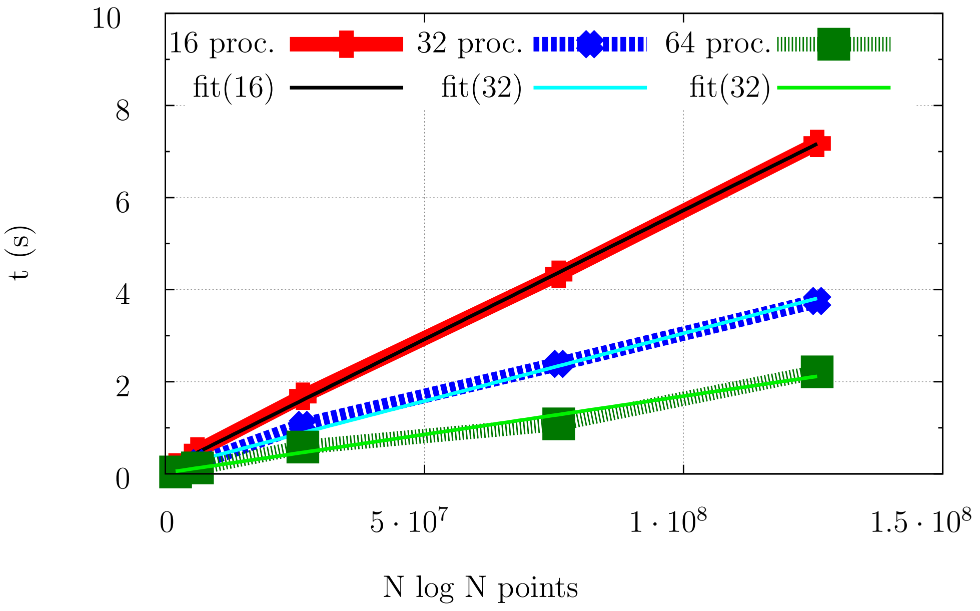
S3 Communication patterns

During the Poisson solver Octopus migh need to use an external library. On those external codes (for example ISF [2] and PFFT [1]) the data distribution migh differ from that on Octopus. Fast Fourier transforms require parallelepipedic grids, but this is not necessarily the case of Octopus. In these cases, the original non-parallelepiped grid has to be converted to a parallelepiped grid by filling the new points with zeroes. At the initialization stage a mapping between the Octopus grid decomposition and the FFT grid decomposition is established and saved. This mapping is used when running the actual solver to efficiently communicate only the strictly necessary grid data between processes, achieving almost perfectly linear parallel scaling.

The PFFT library requires two communication steps in addition to the box transformation. Required communication needs are two `MPI_Alltoall` calls for every calculated FFT. In total, six `MPI_Alltoall` calls are needed in every Poisson solver. However, ISF library is more efficient in this sence and it only needs two `MPI_Alltoall` to calculate the entire Poisson solver; so, in total only four `MPI_Alltoall` are needed.

Regarding to the FMM library, three MPI global communication functions have to be executed: `MPI_Allgather`, `MPI_Allreduce` and `MPI_Alltoall`. Additionally, synchronisation between different FMM levels has to be done using `MPI_Barrier` [4].

A)



B)

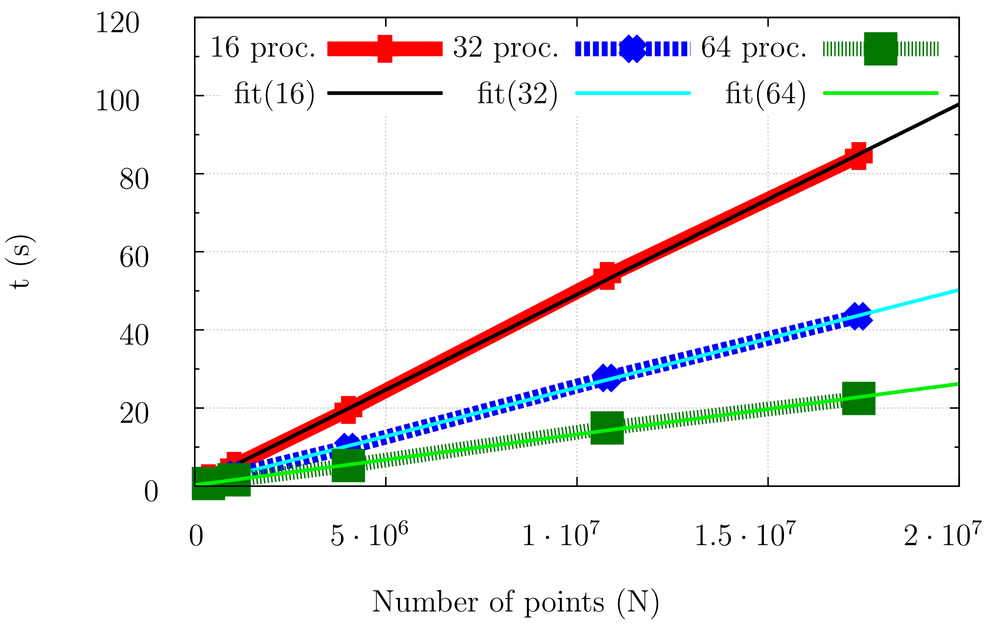


Figure S2: Computational complexities of the PFFT (left) and FMM (right) Poisson solver in the Blue Gene/P supercomputer for a given system. PFFT's fits $\mathcal{O}(N \log N)$, while FMM's its $\mathcal{O}(N)$. PFFT: $fit(16) = 5.6041 \cdot 10^{-8} + 0.1182$, $fit(32) = 2.9407 \cdot 10^{-8} + 0.1149$, and $fit(64) = 1.6656 \cdot 10^{-8} + 0.0228$. FMM: $fit(16) = 4.8760 \cdot 10^{-6} + 0.27730$, $fit(32) = 2.4991 \cdot 10^{-6} + 0.26903$, and $fit(64) = 1.2960 \cdot 10^{-6} + 0.28681$

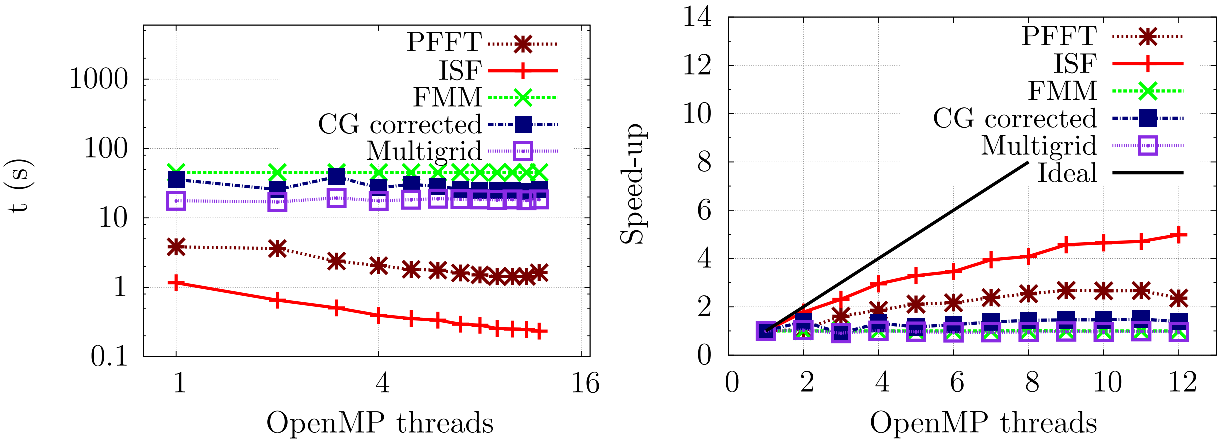


Figure S3: Execution time and speedup for the calculation of the Hartree potential created for a Gaussian charge distribution represented in a cubic grid of edge $L_e = 15.8$ Å and spacing 0.2 Å in a x86-64 architecture (Corvo computer), with one MPI process and from 1 to 12 OpenMP threads.

S4 Tuning the system parameters

In this Section we give additional information on several input parameters that we used in our tests of the Poisson solvers, so that they can be reproduced. These parameters are the spatial form of the grid in whose points the Hartree potential was calculated (`BoxShape`), the accuracy tolerance of the energy calculated with FMM (`DeltaEFMM`), the number of stages in the grid hierarchy of the multigrid solver (`MultigridLevels`) and the order of the multipole expansion of the charges whose potential is analytically calculated when using the multigrid or the conjugate gradient solvers (`PoissonSolverMaxMultipole`).

OCTOPUS is able to handle different grid shapes, like spherical, parallelepiped or *minimum* shapes (the *minimum* mesh shape is the union of spheres centred in each nucleus of the test system). Apart from the mesh shape and size, the grid is defined by its spacing parameter, i.e., the distance between consecutive grid points. The *minimum* mesh shape option in OCTOPUS produces fair results with FMM. Serial FFT, PFFT, and ISF (which uses FFT) solvers create a parallelepiped mesh —which contains the original minimum one— to calculate the Hartree potential. If multigrid or conjugate gradients are used with a minimum mesh, it is frequent to find a serious loss of accuracy, because minimal boxes are usually irregular, and the multipole expansion (whose terms are based on spherical harmonics, which are rather smooth) cannot adapt well to arbitrary charge values in irregular meshes. These effects of box shape made us choose cubic meshes for our tests. In any case, one should take into account that if non-parallelepiped meshes are used, then the solvers based on reciprocal space (serial FFT, PFFT and ISF) spend additional time dealing with the additional points with null density, while FMM does not have this problem. Examples of this can be viewed in Table S1, where it can be observed that the execution time is essentially the same regardless of the box shape for PFFT, while spherical and minimal box shapes are much more efficient than parallelepiped shape for FMM (with a ratio of about 1.67).

Spherical meshes are still valid for multigrid and conjugate gradient methods. We have compared the relative accuracy of cubic and spherical meshes for FMM, multigrid, and CG methods containing the same number of points. When FMM is used, both errors decrease while the mesh size is increased, being relatively equal for both representations, until an accuracy plateau is reached for big meshes. The same behaviour is shown for the multigrid solver and by the CG solver up to a given volume for a given system.

	PFFT			FMM		
$R\|\|L_e$(Å)	Minimal	Sphere	Parallelepiped	Minimal	Sphere	Parallelepiped
15.8	1.001	1.032	1.030	3.371	3.5413	5.945
22.0	2.535	2.664	2.667	10.500	10.447	16.922
25.8	4.321	4.265	4.393	16.321	16.212	27.861
31.6	8.239	8.034	8.271	27.821	28.707	48.206

Table S1: Comparison of total times (s) required for the calculation of the Hartree potential as a function of the box shape and radius (R) or semi-edge length (L_e) for PFFT and FMM solvers.

Apart from their shape (whose effects are commented above) and their size (whose effects are commented in the main paper), the grids are determined by the spacing between points. In Sections 4.1 and 4.2 of the main paper, we showed the accuracy and the execution time of the Poisson solvers as a function of the size (semi-edge) of cubic boxes, all using a spacing of 0.2 Å. This value was chosen because it is close to the values of the spacing typically used in the simulations of physical systems (where, unfortunately, the analytical calculation of the Hartree potential is not possible). The choice of other values of the spacing does not have a strong impact on the results of our tests, as can be viewed in Figure S4. The tests for this Figure used the same Gaussian charge distribution that was defined in the Section 4.1 of the main paper, with a cubic box of semi-edge $L_e = 15.8$ Å and variable spacing. The error Ξ_E as a function of the spacing (see Figure S4A and Figure S4B) is almost invariant within a wide radius around the used value of 0.2. The total number of grid points increases with the cube of spacing^{-1}, and so does (essentially, regardless communication issues) the increasing of the computing time (see Figure S4C). Too big values for the spacing lead to too coarse density samplings, and therefore to wrong Hartree potentials.

The implementations of multigrid and conjugate gradient solvers we used in our tests are based on a multipole expansion. Each value of the input charge density represented on a grid (ρ_{ijk}) is expressed with an analytical term via this multipole expansion, plus a numerical term. The Hartree potential created by the analytic part is calculated analytically, while the Hartree potential created by the numerical term is calculated numerically with either multigrid or conjugate gradient method. So, since the use of the analytic term makes the numerical term smaller, numerical errors are expected to be reduced. Therefore, the smaller the difference between the charge density and its multipole expansion, the smaller the potential numerically calculated, and the higher the accuracy of the total Hartree potential calculation. An order for the multipole expansion (the input parameter `PoissonSolverMaxMultipole`, PSMM) must be chosen. It is to be stressed that arbitrarily higher values of PSMM do not lead to more accurate results; rather, there exists a PSMM that minimises the error for each problem. We ran some tests to obtain this optimal value. In them, we calculated the

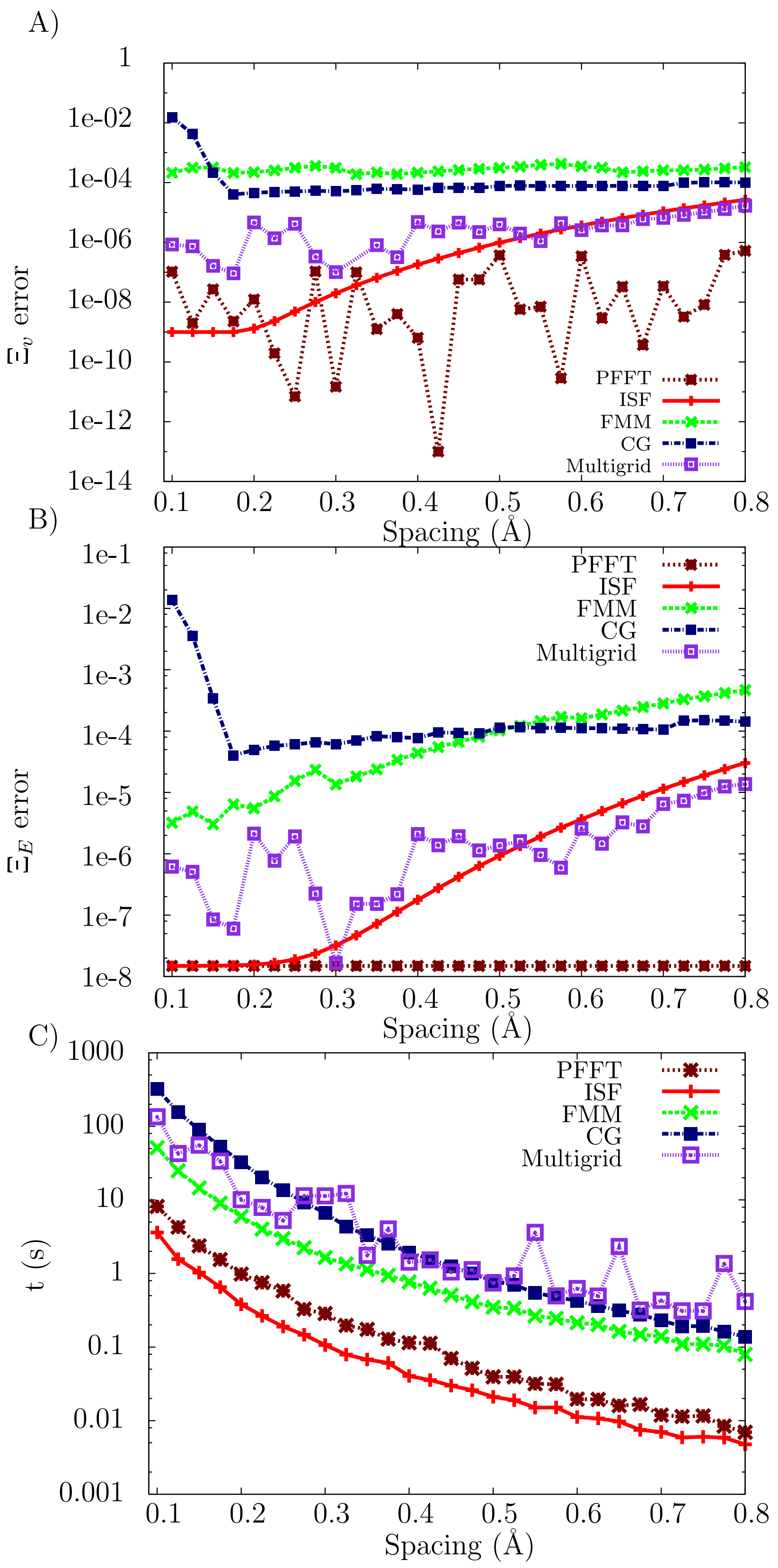


Figure S4: A) Comparison of Ξ_v varying the spacing from 0.1 Å to 0.8 Å, of the system of $L_e = 15.8$ Å. B) Same for Ξ_E error. C) Comparison of time varying the spacing in the same range in Corvo (x86-64). The times are given by running in 8 cores with 8 MPI processes.

ground-state of a 180-atom chlorophyll (see Figure S5) system varying the value of PSMM. Table S2 shows the obtained results; for $PSMM = 8, 9$, and 10, the accumulation of errors was big enough for calculations not to converge (because beyond a given order, the spherical harmonics are too steep to describe the ρ, which is rather smooth). From the data of Table S2, we chose $PSSM = 7$ for our simulations (with $PSSM = 7$ the HOMO-LUMO gap is equivalent to the reference value given by ISF, and the Hartree energy is the closest one).

	Hartree energy (eV)			HOMO-LUMO gap		
PSMM	ISF	CG	Multigrid	ISF	CG	Multigrid
4	240821.47	240813.51	240813.41	1.4489	1.2179	1.2178
6	240821.47	240813.93	240813.95	1.4489	1.4340	1.4353
7	240821.47	240815.01	240814.69	1.4489	1.4520	1.4506

Table S2: Ground state values of the Hartree energy and the HOMO-LUMO gap as a function of the PSMM (input parameter of Multigrid and Conjugate gradients solvers). Reference values are given by the ISF solver.

The implementation we used for FMM [3] allows one to tune the relative error of the calculations. Its expression is the quotient $(E_{ref} - E_n)/E_n$, i.e. the variable `DeltaEFMM`, where E_n is the Hartree energy calculated with the FMM method and E_{ref} is an estimation of what its actual value is. We chose for our calculations a relative error of 10^{-4}. Note that this error corresponds only to the pairwise term of the Hartree potential, before the correction for charge distribution is applied (see Section 2.3 and S7).

S5 Chlorophyll structure

In the Section 4.1 of the paper, we presented the values of some quantities related to observables as calculated with the different analysed solvers (Table 2). The system whose ground state was evaluated for the calculation of these quantities is a chlorophyll stretch consisting of 180 atoms. For clarity's sake, we display the structure of this system in Figure S5.

S6 Correction time of multigrid and conjugate gradients

As explained at the section 2.4 of the main paper, the multigrid and conjugate gradients solvers use a multipolar expansion for the charge, such that the potential created by this expansion can be analytically calculated, while the potential created by the remaining charge, i.e. original charge minus multipolar expansion, is calculated with either multigrid or conjugate gradients. This analytical calculation requires similar times for both solvers. It requires between 0.2% and 8% of the conjugate gradients total execution time, and between 1.7% and 3.2% in the case of multigrid for the standard tests performed. Therefore, in essence, it does not add a significant extra time. As an example, a system of $L_e = 15.8$ and spacing of

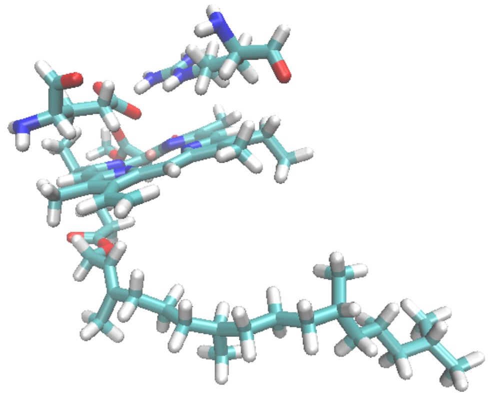


Figure S5: Stretch of a Chlorophyll molecule of the spinach, which was used for the tests of Table 2.

0.2 Å (4,019,679 grid points) in Corvo is shown in Table S3. Very similar trend is shown in other system sizes and machines.

	Multigrid t(s)			CG corrected t(s)		
MPI proc.	Solver	Correction	Percentage	Solver	Correction	Percentage
1	89.02	7.16	8.04%	223.36	7.13	3.19%
2	57.46	3.53	6.14%	122.87	3.58	2.91%
4	29.61	1.87	6.32%	67.72	1.91	2.82%
8	15.60	1.00	6.41%	35.05	1.04	2.96%
16	8.62	0.53	6.20%	20.23	0.53	2.62%
32	4.57	0.28	6.16%	10.87	0.28	2.59%
64	2.84	0.14	4.92%	5.72	0.20	3.42%
128	2.18	0.07	3.17%	4.33	0.08	1.75%
256	5.19	0.08	1.49%	3.46	0.06	1.73%
512	10.04	0.02	0.23%	1.00	0.02	2.33%

Table S3: Correction time (s) compared with the total solver time of multigrid and conjugate gradients solvers in for a system of $L_e = 15.8$ Å (4,019,679 grid points) in Corvo machine from 1 to 512 MPI processes.

S7 Correction terms for FMM

S7.1 General remarks

The fast multipole method (FMM)[5, 6, 7, 8, 3] was devised to efficiently calculate pairwise potentials created by pointlike charges, like pairwise Coulomb potential. In the literature it is possible to find some modifications of the traditional FMM which deal with charges that are modelled as Gaussian functions [9]. Such modifications of FMM can be used into LCAO codes as Gaussian [10] or FHI-Aims. However, they are not useful when the charge distribution is represented through a set of discrete charge density values. The Fast Multipole Method presented in [11, 3] belongs to the family of FMM methods which calculate the electrostatic potential created by a set of pointlike charges. This method is very accurate and efficient, but it needs some modifications to work in programs like OCTOPUS [12, 13], where the 3D grid points actually represent charge densities. As stated in the Section 'Theoretical background' of the main paper, the electronic density is a $\mathbb{R}^3 \to \mathbb{R}$ field, where values of the $\mathbb{R}^3$ set correspond to an equispaced grid (see Figure S6C)). The variable $\rho_{j,k,l}$ is the charge density at the portion of volume (cell) centred in the point (j, k, l). Each cell is limited by the planes bisecting the lines that join two consecutive grid points, and its volume is $\Omega = h^3$, being h the spacing between consecutive grid points. The density $\rho_{j,k,l}$ is always negative and it is expected to vary slowly among nearby points.

The term v^{SI} in equation (16) of the main paper can be calculated analytically as follows assuming that the cell is a sphere of volume Ω:

$$\begin{aligned} v^{\mathrm{SI}}(\vec{r_0}) &= \int_\Omega d\vec{r}\frac{\rho(\vec{r})}{|\vec{r}-\vec{r_0}|} = \rho(\vec{r_0})\int_\Omega d\vec{r}\frac{1}{|\vec{r}-\vec{r_0}|} \\ &\simeq \rho(\vec{r_0})\int_0^R dr\int_0^{2\pi} d\phi\int_0^\pi d\theta\frac{r^2 sen(\theta)}{r} \\ &= \rho(\vec{r_0})2\pi h^2\left(\frac{3}{4\pi}\right)^{2/3}, \end{aligned} \tag{1}$$

where we have used the approximation of constant charge density within the cell. One may expect this approximate way to proceed to be less accurate than the numerical integration of $1/r$ in a cubic cell (what is also efficient, since the integration through an arbitrary size cube is proportional to the integral through a cube of unit volume). However, it happens the converse: the difference between both methods is small (about 1% of difference between integrals) but, due to error cancellations, the analytical method is slightly more accurate when calculating potentials.

The term $v^{\mathrm{corr.}}_{j,k,l}$ in (16) is included to calculate more accurately the potential created by the charge in cells nearby to (j, k, l). To devise a expression for it, we consider that the charge distribution is similar to sets of Gaussians centred in the atoms of the system. For Gaussian distributions, the greatest concavity near the centre of the Gaussian makes the influence of neighbouring points to be major for the potential. As we can see in the scheme of Figure S6 A)-B), considering semi-neighbours of point P (in $\vec{r_0} := (j, k, l)$), i.e., points whose distance to P is not h, but $h/2$, the integral of equation 16 of the paper can be calculated in a much more accurate way.

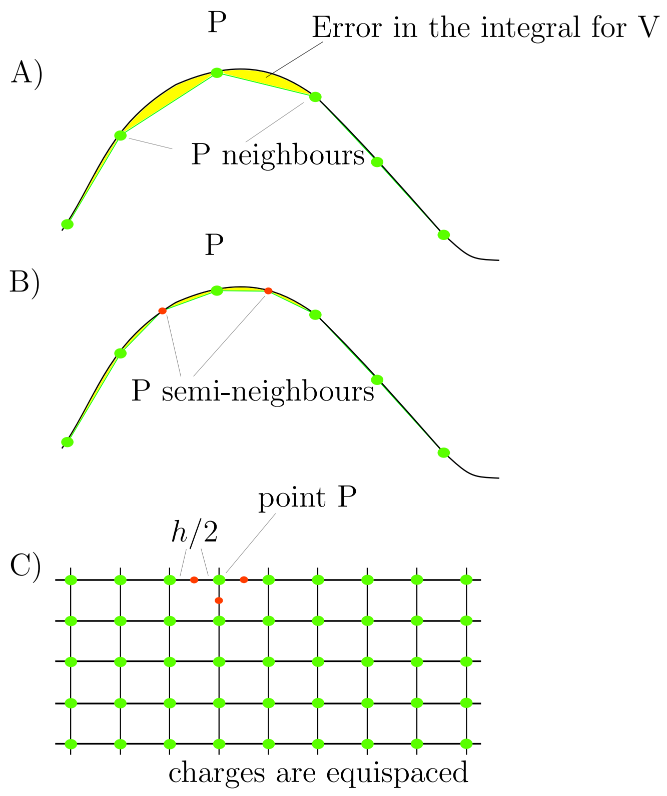


Figure S6: Scheme of how the inclusion of semi-neighbours of point P (pink points) helps to improve the accuracy of the integration to calculate the Hartree potential. A) Scheme of the function whose integration will be approximated by a summation (surface under the green line), without considering semi-neighbours. B) id., considering semi-neighbours of point P; The error made by the approximation is proportional to the yellow surface in A) and B). C) 2D scheme of the grid: green points are grid points, while pink points are semi-neighbours of P.

S7.2 Method 1: 6-neighbours correction

We build a corrective term by calculating the charge in the 6 semi-neighbours of every point of the mesh $\vec{r_0}$ (see Figure S7 for a intuitive scheme). The total correction term is the potential created by the semi-neighbours ($v^{\text{corr.+}}$) minus the potential created by the charge lying in the volume of the semi-neighbour cells that was already counted in v^{FMM} or in v^{SI} ($v^{\text{corr.}-}$):

$$v^{\text{corr.}}(\vec{r_0}) = v^{\text{corr.+}}(\vec{r_0}) - v^{\text{corr.}-}(\vec{r_0}) \ . \tag{2}$$

In order to calculate $v^{\text{corr.}+}$, we use the formula por the 3rd degree interpolation polynomial:

$$f(0) = \frac{(-1)}{16} f\left(\frac{-3}{2}h\right) + \frac{9}{16} f\left(\frac{-h}{2}\right) + \frac{9}{16} f\left(\frac{h}{2}\right) - \frac{(-1)}{16} f\left(\frac{3}{2}h\right) \ , \tag{3a}$$

$$f\left(\frac{-h}{2}\right) = \frac{(-1)}{16} f(-2h) + \frac{9}{16} f(-h) + \frac{9}{16} f(0) - \frac{(-1)}{16} f(h) \ , \tag{3b}$$

$$f\left(\frac{h}{2}\right) = \frac{(-1)}{16} f(-h) + \frac{9}{16} f(0) + \frac{9}{16} f(h) - \frac{(-1)}{16} f(2h) \ . \tag{3c}$$

So, the semi-neighbours of $\vec{r_0} = (x_0, y_0, z_0)$ are

$$\begin{aligned} \rho(x_0 - h/2, y_0, z_0) =& \frac{-1}{16}\rho(x_0 - 2h, y_0, z_0) + \frac{9}{16}\rho(x_0 - h, y_0, z_0) \\ &+ \frac{9}{16}\rho(x_0, y_0, z_0) - \frac{1}{16}\rho(x_0 + h, y_0, z_0) \ ; \end{aligned} \tag{4a}$$

$$\begin{aligned} \rho(x_0 + h/2, y_0, z_0) =& \frac{-1}{16}\rho(x_0 - h, y_0, z_0) + \frac{9}{16}\rho(x_0, y_0, z_0) \\ &+ \frac{9}{16}\rho(x_0 + h, y_0, z_0) - \frac{1}{16}\rho(x_0 + 2h, y_0, z_0) \ ; \end{aligned} \tag{4b}$$

$$\begin{aligned} \rho(x_0, y_0 - h/2, z_0) =& \frac{-1}{16}\rho(x_0, y_0 - 2h, z_0) + \frac{9}{16}\rho(x_0, y_0 - h, z_0) \\ &+ \frac{9}{16}\rho(x_0, y_0, z_0) - \frac{1}{16}\rho(x_0, y_0 + h, z_0) \ ; \end{aligned} \tag{4c}$$

$$\begin{aligned} \rho(x_0, y_0 + h/2, z_0) =& \frac{-1}{16}\rho(x_0, y_0 - h, z_0) + \frac{9}{16}\rho(x_0, y_0, z_0) \\ &+ \frac{9}{16}\rho(x_0, y_0 + h, z_0) - \frac{1}{16}\rho(x_0, y_0 + 2h, z_0) \ ; \end{aligned} \tag{4d}$$

$$\begin{aligned} \rho(x_0, y_0, z_0 - h/2) =& \frac{-1}{16}\rho(x_0, y_0, z_0 - 2h) + \frac{9}{16}\rho(x_0, y_0, z_0 - h) \\ &+ \frac{9}{16}\rho(x_0, y_0, z_0) - \frac{1}{16}\rho(x_0, y_0, z_0 + h) \ ; \end{aligned} \tag{4e}$$

$$\begin{aligned} \rho(x_0, y_0, z_0 + h/2) =& \frac{-1}{16}\rho(x_0, y_0, z_0 - h) + \frac{9}{16}\rho(x_0, y_0, z_0) \\ &+ \frac{9}{16}\rho(x_0, y_0, z_0 + h) - \frac{1}{16}\rho(x_0, y_0, z_0 + 2h) \ . \end{aligned} \tag{4f}$$

We consider all this six charges to be homogeneously distributed in cells whose volume is $\Omega/8$. The distance between the centre of these small cells and the centre of the cell whose v we are calculating (i.e., the cell centred in $\vec{r_0}$) is $h/2$. So the first part of $v^{\text{corr.}}(\vec{r_0})$ is

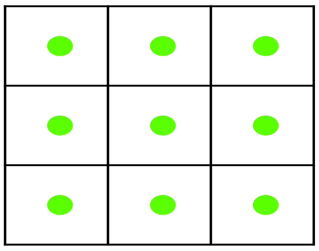
$$\begin{aligned} v^{\text{corr.}+}(\vec{r_0}) \ &= \ \Big(\rho(x_0 - h/2, y_0, z_0) + \rho(x_0 + h/2, y_0, z_0) + \ldots + \\ &+ \ \rho(x_0, y_0, z_0 + h/2)\Big) \left(\frac{\Omega}{8}\right) \left(\frac{1}{h/2}\right) \ . \end{aligned} \tag{5}$$

Since we have created these new 6 cells, we must subtract the potential created by their corresponding volume from that created by the cells whose volume is partly occupied by these new cells. This potential is:

$$
\begin{aligned}
v^{\text{corr.}-}(\vec{r_0}) \;=\; & \Big(\rho(x_0-h,y_0,z_0)+\rho(x_0+h,y_0,z_0)+\rho(x_0,y_0-h,z_0)+\rho(x_0,y_0+h,z_0) \\
& +\; \rho(x_0,y_0,z_0-h)+\rho(x_0,y_0,z_0+h)\Big)\left(\frac{\Omega}{16}\right)\left(\frac{1}{h}\right)+\alpha\ v^{\text{SI}}(\vec{r_0})\ . \qquad (6)
\end{aligned}
$$

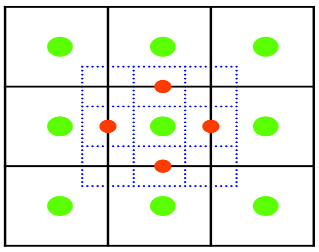
The aim of the term $\alpha\ v^{\text{SI}}$ (i.e., the variable `AlphaFMM` in OCTOPUS) is to compensate the errors arising from the assumption that the charge is concentrated at the centre of the cells and reduced cells. The value of α is tuned to minimize the errors in the potentials.

A)



Original box: all cell's size is h^2 (2D)

B)



New: Semi-neighbours cell sizes are $h^2/4$

Side cells sizes are $7/8h^2$

Central cell size is $h^2/2$

(all in 2D)

Figure S7: 2D example of the position of cells containing semi neighbours. Assume the centre of the plots is $\vec{r_0}$, the point where we want to calculate the correcting term for the potential. The volume of semi-neighbour cells is $h^2/4$ in 2D, and $\Omega/8$ in 3D. One half of the semi-neighbour cell occupies the volume of a neighbour cell (the cell whose centre is h away from $\vec{r_0}$). The other half of the semi-neighbour cell occupies the space of the $\vec{r_0}$-centred cell itself.

It is worth to re-express as follows the correction terms of eqs. (2), (5) and (6) avoiding to call to every variable more than once for the sake of getting higher computational efficiency:

$$
\begin{aligned}
v^{\text{SI}}(\vec{r_0})+v^{\text{corr.}}(\vec{r_0}) = h^2\Big[\quad & \rho(x_0,y_0,z_0)\big(27/32+(1-\alpha)2\pi(3/4\pi)^{2/3}\big) \qquad (7) \\
& +(1/16)\Big(\rho(x_0-h,y_0,z_0)+\rho(x_0+h,y_0,z_0)+\rho(x_0,y_0-h,z_0) \\
& \quad +\rho(x_0,y_0+h,z_0)+\rho(x_0,y_0,z_0-h)+\rho(x_0,y_0,z_0+h) \\
& \quad -(1/4)\big(\rho(x_0-2h,y_0,z_0)+\rho(x_0+2h,y_0,z_0)+\rho(x_0,y_0-2h,z_0) \\
& \quad +\rho(x_0,y_0+2h,z_0)+\rho(x_0,y_0,z_0-2h)+\rho(x_0,y_0,z_0+2h)\big)\Big)\Big]\ .
\end{aligned}
$$

We ran tests using the error formula $E := \sqrt{\sum_i (v^{\text{Exact}}(\vec{r_i}) - v^{\text{FMM}}(\vec{r_i}))^2}$, with the index i running for all points of the system. The inclusion of the correcting term introduced in this Section typically reduced E in a factor about 50.

S7.3 Method 2: 124-neighbours correction

This method is similar to the one explained in the previous Section, but with two differences

- It uses 3D interpolation polynomials, instead of 1D polynomials. Then, it considers $5^3 - 1 = 124$ neighbours in a cube of edge $5h$ centred in $\vec{r_0}$ to calculate the corrective term for $V(\vec{r_0})$
- The interpolation polynomials representing $\rho(x, y, z)$ are numerically integrated (after their division by r). This is, we calculate

$$v^{\text{corr.+}}(\vec{r_0}) \;=\; \int_{125\Omega} d\vec{r} \frac{\rho(\vec{r})}{|\vec{r} - \vec{r_0}|} \simeq \int_{125\Omega} d\vec{r} \frac{Pol(\vec{r})}{|\vec{r} - \vec{r_0}|} \;. \tag{8}$$

The integration is to be performed between -5/2h and 5/2h for x, y and z. The interpolation polynomial (with 125 support points) $Pol(\vec{r})$ is

$$Pol(\vec{r}) = \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5} \rho(x_0 + (i-3)h, y_0 + (j-3)h, z_0 + (k-3)h)\alpha_i(x)\alpha_j(y)\alpha_k(z) \;, \tag{9}$$

being

$$\alpha_1(\xi) := \frac{\xi^4}{24} - \frac{\xi^3}{12} - \frac{\xi^2}{24} + \frac{\xi}{12} \;, \tag{10a}$$

$$\alpha_2(\xi) := -\frac{\xi^4}{6} + \frac{\xi^3}{6} + \frac{2\xi^2}{3} - \frac{2\xi}{3} \;, \tag{10b}$$

$$\alpha_3(\xi) := \frac{\xi^4}{4} - \frac{5\xi^2}{4} + 1 \;, \tag{10c}$$

$$\alpha_4(\xi) := -\frac{\xi^4}{6} - \frac{\xi^3}{6} + \frac{2\xi^2}{3} + \frac{2\xi}{3} \;, \tag{10d}$$

$$\alpha_5(\xi) := \frac{\xi^4}{24} + \frac{\xi^3}{12} - \frac{\xi^2}{24} - \frac{\xi}{12} \;. \tag{10e}$$

The quotient of the polynomials $\alpha_i(x)\alpha_j(y)\alpha_k(z)$ divided by $|\vec{r} - \vec{r_0}|$ can be numerically integrated through the cubic cell of edge $5h$ and centred in x_0. Such integrals can indeed be tabulated, because equation (8) implies $v^{\text{corr.+}}(\vec{r_0}) = v^{\text{corr.+}}(\vec{r_0})|_{h=1} \cdot h^2$. Terms of $\alpha_i(x)\alpha_j(y)\alpha_k(z)/|\vec{r} - \vec{r_0}|$ are often odd functions whose integral is null. The non-zero integrals taking part in (1) (with $h = 1$) can be easily calculated numerically.

Therefore

$$
\begin{aligned}
v^{\text{corr.}+}(\vec{r_0}) &= \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\cdot \\
&\quad \int_{125\Omega} d\vec{r}\frac{\alpha_i(x)\alpha_j(y)\alpha_k(z)}{|\vec{r}-\vec{r_0}|} \\
&= \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\cdot \\
&\quad \sum_{l=1}^{5}\sum_{m=1}^{5}\sum_{n=1}^{5}\alpha_{i,l}\alpha_{j,m}\alpha_{k,n}\int_{125\Omega} d\vec{r}\frac{x^{l-1}y^{m-1}z^{n-1}}{|\vec{r}-\vec{r_0}|} \\
&= \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\cdot \\
&\quad \sum_{l=1}^{5}\sum_{m=1}^{5}\sum_{n=1}^{5}\alpha_{i,l}\alpha_{j,m}\alpha_{k,n}\beta(l-1,m-1,n-1)h^2 \ , \qquad (11)
\end{aligned}
$$

where $\alpha_{i,l}$ is the coefficient of ξ^{l-1} if $\alpha_i(\xi)$ and

$$\beta(l,m,n) := \int_{-1/2}^{1/2}dx\int_{-1/2}^{1/2}dy\int_{-1/2}^{1/2}dz\frac{x^l y^m z^n}{\sqrt{x^2+y^2+z^2}} \ . \qquad (12)$$

In this case, $v^{\text{corr.}-}$ is equal to all the contributions to $V(\vec{r_0})$ due to charges whose position (x, y, z) satisfies

$$|x-x_0| <= 2h; \ |y-y_0| <= 2h; \ |z-z_0| <= 2h \ , \qquad (13)$$

including self-interaction integral.

This way to calculate $v^{\text{corr.}+}$ is not inefficient, because only 27 integrals are not null, and both α and β are known. In order to calculate $v^{\text{corr.}+}(\vec{r_0})$ we need 125 products and additions, what is essentially the same number of operations which is required in order to calculate the potential created in $\vec{r_0}$ by the neighbouring points (whose calculation can be removed and then saved). Nevertheless, results using this correction method were worse than that obtained using the first method, so only that one was implemented into the standard version of OCTOPUS.

References

[1] M. Pippig, *PFFT - An Extension of FFTW to Massively Parallel Architectures*, SIAM J. Sci. Comput. **35** (2013) C213 – C236.

[2] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, and G. Beylkin, *Efficient solution of Poisson's equation with free boundary conditions*, J. Chem. Phys. **125** (2006) 074105, Genovese, L. and Deutsch, T. and Neelov, A. and Goedecker, S. and Beylkin, G.

[3] H. Dachsel, *An error controlled fast multipole method*, J. Chem. Phys. **132** (2010) 119901.

[4] I. Kabadshow, H. Dachsel, and J. Hammond, *Poster: Passing the three trillion particle limit with an error-controlled fast multipole method*, in *Proceedings of the 2011 companion on High Performance Computing Networking, Storage and Analysis Companion*, SC '11 Companion, pp. 73–74, New York, NY, USA, 2011, ACM.

[5] L. F. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys. **73** (1987) 325–348.

[6] L. F. Greengard and V. Rokhlin, *The Rapid Evaluation of Potential Fields in Three Dimensions*, Springer Press, Berlin, Heidelberg, 1988.

[7] L. F. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numerica **6** (1997) 229.

[8] H. Cheng, L. F. Greengard, and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, J. Comp. Phys **155** (1999) 468–498.

[9] M. Strain, G. Scuseria, and M. Frisch, *Achieving Linear Scaling for the Electronic Quantum Coulomb Problem*, Science **271** (1996) 51–53.

[10] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, *Gaussian 09 Revision A.1*, 2009, Gaussian Inc. Wallingford CT 2009.

[11] I. Kabadshow and H. Dachsel, *The Error-Controlled Fast Multipole Method for Open and Periodic Boundary Conditions*, in *Fast Methods for Long-Range Interactions in Complex Systems*, IAS Series, Volume 6, Forschungszentrum Jülich, Germany, 2010, CECAM.

[12] A. Castro, H. Appel, M. Oliveira, C. Rozzi, X. Andrade, F. Lorenzen, M. Marques, E. Gross, and A. Rubio, *Octopus: a tool for the application of time-dependent density functional theory*, Phys. Stat. Sol. B **243** (2006) 2465–2488, The

OCTOPUS code is available in a public Subversion repository `http://www.tddft.org/svn/octopus/trunk`.

[13] M. A. L. Marques, A. Castro, G. F. Bertsch, and A. Rubio, *octopus: a first-principles tool for excited electron-ion dynamics*, Computer Physics Communications **151** (2003) 60.